



**NAVAL
POSTGRADUATE
SCHOOL**

MONTEREY, CALIFORNIA

DISSERTATION

**UNSCENTED SAMPLING TECHNIQUES FOR
EVOLUTIONARY COMPUTATION WITH APPLICATIONS
TO ASTRODYNAMIC OPTIMIZATION**

by

Christopher B. McGrath

September 2016

Dissertation Supervisor:

I. Michael Ross

Approved for public release. Distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
<p><i>Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.</i></p>			
1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE 09-23-2016	3. REPORT TYPE AND DATES COVERED Dissertation 09-21-2013 to 09-23-2016	
4. TITLE AND SUBTITLE UNSCENTED SAMPLING TECHNIQUES FOR EVOLUTIONARY COMPUTATION WITH APPLICATIONS TO ASTRODYNAMIC OPTIMIZATION		5. FUNDING NUMBERS	
6. AUTHOR(S) Christopher B. McGrath			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943		8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A		10. SPONSORING / MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this document are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol Number: N/A.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.		12b. DISTRIBUTION CODE	
13. ABSTRACT (maximum 200 words) This dissertation investigates several innovative approaches to evolutionary optimization that are relevant to numerous applications in astronautical engineering. The challenges and shortfalls associated with evolutionary algorithms are translated into three overarching goals that directly motivate the research and innovations of this dissertation. The first goal is to investigate and employ techniques that enable evolutionary algorithms to effectively handle constraints in a way that allows for feasible solutions to constrained optimization problems. The second goal is to improve computation times and efficiencies associated with evolutionary algorithms. The last goal is to enhance the evolutionary algorithm's robustness and ability to consistently find accurate solutions within a finite number of iterations. Novel techniques involving the application of unscented sampling, parallel computation, and various forms of exact penalty functions are developed and applied to both genetic algorithms and evolution strategies to achieve these goals. The results of this research offer a promising new set of modified evolutionary algorithms that outperform state-of-the-art techniques on a number of challenging multimodal optimization problems. In addition, these new methods are shown to be very effective in solving a minimum-propellant lunar lander optimal control problem, representing a class of problems that are historically difficult to solve using evolutionary algorithms.			
14. SUBJECT TERMS evolutionary algorithm, evolution strategy, genetic algorithm, parallel computation, parallel processing, unscented sampling		15. NUMBER OF PAGES 417 16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

**UNSCENTED SAMPLING TECHNIQUES FOR EVOLUTIONARY
COMPUTATION WITH APPLICATIONS TO ASTRODYNAMIC
OPTIMIZATION**

Christopher B. McGrath
Captain, United States Air Force

B.S., Astronautical Engineering, United States Air Force Academy, 2007
M.S., Aerospace Engineering, California State Polytechnic University, 2009

Submitted in partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY IN ASTRONAUTICAL ENGINEERING
from the
NAVAL POSTGRADUATE SCHOOL
September 2016

Approved by:	I. Michael Ross, Ph.D. Professor of Mechanical and Aerospace Engineering Dissertation Supervisor	Mark Karpenko, Ph.D. Research Associate Professor of Mechanical and Aerospace Engineering
	Ronald Proulx, Ph.D. Research Professor of Space Systems Engineering	Isaac Kaminer, Ph.D. Professor of Mechanical and Aerospace Engineering
	Wei Kang, Ph.D. Professor of Applied Mathematics	Lucas Wilcox, Ph.D. Associate Professor of Applied Mathematics
Approved by:	Garth Hobson, Ph.D. Chair, Department of Mechanical and Aerospace Engineering	
Approved by:	Douglas Moses, Ph.D. Vice Provost for Academic Affairs	

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This dissertation investigates several innovative approaches to evolutionary optimization that are relevant to numerous applications in astronautical engineering. The challenges and shortfalls associated with evolutionary algorithms are translated into three overarching goals that directly motivate the research and innovations of this dissertation. The first goal is to investigate and employ techniques that enable evolutionary algorithms to effectively handle constraints in a way that allows for feasible solutions to constrained optimization problems. The second goal is to improve computation times and efficiencies associated with evolutionary algorithms. The last goal is to enhance the evolutionary algorithm's robustness and ability to consistently find accurate solutions within a finite number of iterations. Novel techniques involving the application of unscented sampling, parallel computation, and various forms of exact penalty functions are developed and applied to both genetic algorithms and evolution strategies to achieve these goals. The results of this research offer a promising new set of modified evolutionary algorithms that outperform state-of-the-art techniques on a number of challenging multimodal optimization problems. In addition, these new methods are shown to be very effective in solving a minimum-propellant lunar lander optimal control problem, representing a class of problems that are historically difficult to solve using evolutionary algorithms.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

1 Optimization and Optimal Control	1
1.1 Introduction	1
1.2 Several Optimization Problems in Astrodynamics	2
1.3 Optimal Control Techniques and Associated Challenges	8
1.4 No Free Lunch Theorem For Optimization Algorithms	16
1.5 Dissertation Research Contributions	17
1.6 Dissertation Overview	20
2 Parallel Computing Concepts	25
2.1 Introduction	25
2.2 Shift Toward Mainstream Parallel Processing	26
2.3 Hardware	27
2.4 Program Architectures	30
2.5 Parallel Footprint Generation For RLVs	31
2.6 Generating RLV Footprints	35
2.7 Parallel RLV Footprint Program Architecture	41
2.8 Parallel RLV Footprint Determination Results	44
2.9 Conclusion	55
3 Genetic Algorithms	57
3.1 Introduction	57
3.2 Fitness Function.	59
3.3 Initial Population	61
3.4 Selection Operators	61
3.5 Encoding	65
3.6 Crossover	68
3.7 Mutation	71
3.8 Stopping Criteria	71
3.9 Schema Theory	73

3.10 Building Block Concept	76
3.11 Deceptive Problems	76
3.12 Dynamic Modeling and Convergence of Genetic Algorithms	79
3.13 Genetic Algorithm Parameter Settings	86
3.14 Binary Genetic Algorithm Improvements	92
3.15 Real Coded Genetic Algorithms	95
3.16 Useful Takeaways From GA and RCGA Literature Review	107
3.17 Parallel Genetic Algorithm Architectures	108
3.18 Conclusion	115
4 Parallel Genetic Algorithms and Optimal Control	117
4.1 Introduction	117
4.2 Lunar Lander Problem	119
4.3 Direct Shooting	120
4.4 Constraint Handling	123
4.5 An Overview of Genetic Algorithms Used	138
4.6 Experimental GA Results	146
4.7 Conclusion	154
5 Evolution Strategies	157
5.1 Introduction	157
5.2 Conventional Evolution Strategy	158
5.3 ES Naming Convention	160
5.4 $(1 + 1)$ ES	160
5.5 $(\mu + 1)$ ES	161
5.6 $(\mu^+ \lambda)$ ES	162
5.7 $(\mu/\rho^+ \lambda)$ ES	163
5.8 Mutation	164
5.9 Selection	171
5.10 Recombination Techniques	171
5.11 Covariance Adaptation Techniques	175
5.12 CMA-ES	180

5.13	NES	183
5.14	Coordinate Invariant xNES	189
5.15	ES Theory	192
5.16	Conclusion	194

6 Unscented Sampling In ESs **195**

6.1	Introduction	195
6.2	RSA-ES	196
6.3	USA-ES	197
6.4	Benchmark Test Problems	201
6.5	Recombination Trade Studies	203
6.6	USA-ES vs. RSA-ES Comparison	211
6.7	USA-ES Local Optima Analysis	220
6.8	USA-ES vs. CMA-ES Comparison	224
6.9	GHSA-ES	231
6.10	Discussion and Related Studies	237
6.11	Application of USA-ES To Trajectory Optimization.	239
6.12	Conclusion	244

7 Unscented Sampling In NESs **247**

7.1	Introduction	247
7.2	uNES	248
7.3	Accuracy of $\nabla_{\mu}J(\mu, \sigma)$ Search Gradient Estimates	251
7.4	Accuracy of $\nabla_{\sigma}J(\mu, \sigma)$ Search Gradient Estimates	263
7.5	uxNES	271
7.6	Performance of uxNES	281
7.7	Learning Rates	285
7.8	xNES/uxNES/USA-ES Comparison on Test Functions	289
7.9	Global Optimization Algorithm Using uxNES	293
7.10	uxNES Lunar Lander Optimal Control Application	296
7.11	Conclusion	298

8 Conclusions and Future Work	299
8.1 Introduction	299
8.2 Contributions and Relevancy.	299
8.3 Relevant Astrodynamical Applications	301
8.4 Further Development of the Unscented RCGA.	304
8.5 Further Development of the USA-ES	305
8.6 Further Development of The uxNES.	307
8.7 Development and Application of The Unscented CMA-ES	308
8.8 Implementation of Unscented Sampling In Other Stochastic Algorithms . . .	309
8.9 Conclusion.	314
Appendix A Definition of Symbols By Chapter	315
A.1 Chapter 1 Symbol Definitions	315
A.2 Chapter 2 Symbol Definitions	315
A.3 Chapter 3 Symbol Definitions	317
A.4 Chapter 4 Symbol Definitions	320
A.5 Chapter 5 Symbol Definitions	322
A.6 Chapter 6 Symbol Definitions	325
A.7 Chapter 7 Symbol Definitions	326
A.8 Chapter 8 Symbol Definitions	327
A.9 Appendix B Symbol Definitions	328
A.10 Appendix C symbol definitions	330
Appendix B Survey of Constraint Handling Techniques For Evolutionary Algorithms	331
B.1 Penalty Functions	332
B.2 Genetic Operator Techniques.	344
B.3 Repair Methods	347
B.4 Co-evolution and Multiple Objective Optimization	348
B.5 Mapping and Decoder Methods	349
B.6 Constraint Sequencing	350
B.7 Summary	350

Appendix C An Unscented Genetic Algorithm	353
C.1 Unscented Crossover Operator	353
C.2 Unscented GA Lunar Lander Optimal Control Application	354
List of References	359
Initial Distribution List	383

THIS PAGE INTENTIONALLY LEFT BLANK

List of Figures

Figure 1.1	An oversubscribed commercial imaging satellite problem	6
Figure 1.2	A multi-loop hybrid optimal control approach	7
Figure 2.1	Visualization of pipelining	28
Figure 2.2	Example footprint points generated using weighted cost function method	37
Figure 2.3	Example footprint points using latitude sweep method	38
Figure 2.4	Example footprint points using longitude sweep method	39
Figure 2.5	Example footprint points using angle sweep method	40
Figure 2.6	Method 1 RLV parallel program model (MATLAB Parallel Toolbox)	43
Figure 2.7	Method 2 RLV parallel program model (MPI shell script)	44
Figure 2.8	48 point benchmark RLV footprint	45
Figure 2.9	Solution times for each point on the benchmark RLV footprint . .	45
Figure 2.10	Example partitioned RLV footprint for scaling	47
Figure 2.11	Speedup results for method 1 - shared memory (48 point footprint)	49
Figure 2.12	Speedup results for method 2 - shared memory (48 point footprint)	50
Figure 2.13	Work distribution for solving a footprint using 8 cores	51
Figure 2.14	Time-based work distribution for solving a footprint using 2 cores	52
Figure 2.15	Time-based work distribution for solving a footprint using 4 cores	53
Figure 2.16	Time-based work distribution for solving a footprint using 8 cores	53
Figure 2.17	Shared memory coherency penalty curve	54
Figure 2.18	Speedup comparison for method 2 comparing workload balancing and memory models	55

Figure 3.1	An illustration of a simple GA	59
Figure 3.2	A comparison of a standard algorithm and a GA	61
Figure 3.3	An illustration of a roulette selection operator	63
Figure 3.4	A diagram of a tournament selection operator (minimization) . .	64
Figure 3.5	A single-point crossover operator	69
Figure 3.6	A multi-point crossover operator	69
Figure 3.7	A uniform crossover operator	70
Figure 3.8	A bit-wise mutation operator	71
Figure 3.9	An example simulated binary crossover (SBX) probability density function	102
Figure 3.10	An example fuzzy recombination probability density function . .	102
Figure 3.11	A master-follower parallel GA model	109
Figure 3.12	Theoretical speedup curves for a master-follower GA	111
Figure 3.13	A round-robin parallel island GA model	112
Figure 3.14	A hybrid parallel GA model	114
Figure 4.1	Venn diagram of parallel GA optimal control study	118
Figure 4.2	Illustration of feasible region for penalty function problem 1 . .	126
Figure 4.3	Effects of penalty parameter value on solution feasibility for problem 1	129
Figure 4.4	Average effects of penalty function type and parameter value for problem 1	131
Figure 4.5	Average effects of penalty function type and parameter value for problem 2	133
Figure 4.6	Illustration of equality formulation vs. inequality formulation for parallel and serial GAs	135
Figure 4.7	Crossover/mutation truncated Gaussian	142

Figure 4.8	Parallel island architecture (round-robin)	145
Figure 4.9	Lunar lander adaptive penalty using dynamic truncated Gaussian crossover (DTGX)	147
Figure 4.10	Effects of number of processors on GA performance over 10^6 generations	148
Figure 4.11	Effects of number of migrations (per 10^6 generations) on GA performance	150
Figure 4.12	Comparison of GA types for lunar lander problem	152
Figure 4.13	Arithmetic RCGA 50-node lunar lander trajectory: parallel(a) vs. serial(b)	154
Figure 5.1	Standard ES algorithm	159
Figure 5.2	(1+1) ES illustration	161
Figure 5.3	Single parent/multiple offspring ES illustration	162
Figure 5.4	Model of a $(\mu + \lambda)$ ES	163
Figure 5.5	Model of a $(\mu/\rho + \lambda)$ ES	164
Figure 5.6	Spherical mutation illustration	167
Figure 5.7	Axis-parallel mutation illustration	168
Figure 5.8	Full covariance mutation illustration	170
Figure 5.9	Dominant (discrete) recombination technique	173
Figure 5.10	Logarithmic weighting scheme ($\rho = 10$)	175
Figure 6.1	Illustration of 3-D 3rd order sigma points with varying κ values .	200
Figure 6.2	Unreachable zones with parameterization of Gaussian	201
Figure 6.3	3-D Visualization Of Benchmark Problems	203
Figure 6.4	Recombination method comparison on “shallow” 10-D functions	205
Figure 6.5	Recombination method comparison on “steep” 10-D function .	206

Figure 6.6	ρ comparison on “shallow” 10-D functions	208
Figure 6.7	ρ comparison on “steep” 10D function	209
Figure 6.8	Average USA-ES/RSA-ES fitness progression on 40-D test functions with equal number of sampling points	215
Figure 6.9	Average USA-ES/RSA-ES fitness progression as sample points and function evaluations are increased	219
Figure 6.10	Average USA-ES/RSA-ES fitness progression over 100 runs as dimension is increased	220
Figure 6.11	Numerical gradient limit tests	222
Figure 6.12	USA-ES gradient L2 norm progression on 10-D test functions over a single run	224
Figure 6.13	Average USA-ES/CMA-ES fitness progression on 40-D test functions with equal number of sampling points	227
Figure 6.14	Average USA-ES/CMA-ES fitness progression on 40-D test functions with unequal numbers of sampling points	230
Figure 6.15	Gauss-Hermite point growth w.r.t. accuracy level and dimension using Smolyak sparse grids	232
Figure 6.16	Comparison of parameterization methods	233
Figure 6.17	USA-ES vs. GHSA-ES on 40D test functions	235
Figure 6.18	Simple round-robin parallel island architecture	242
Figure 6.19	USA-ES vs. RSA-ES lunar lander problem average fitness progression	243
Figure 6.20	USA-ES solutions to the lunar lander problem	244
Figure 7.1	Accuracy of $\nabla_\mu J(\mu, \sigma)$ on the 1-D functions $f(x) = x^2$ and $f(x) = x^9$ ($\mu = 0$, $\sigma = 1$)	254
Figure 7.2	Accuracy of $\nabla_\mu J(\mu, \sigma)$ on the 1-D function $f(x) = x^5$ ($\mu = 1$ and varying σ)	258

Figure 7.3	Accuracy of $\nabla_\mu J(\mu, \sigma)$ on the 1-D function $f(x) = x^5$ ($\sigma = 0.1$ and varying μ)	260
Figure 7.4	$\nabla_\mu J(\mu, \sigma)$ accuracy for the 10-D function $f(\bar{x}) = \sum_{i=1}^n x_i^5$ ($\bar{\mu} = [1, \dots, n]^T$ and $C = 0.1 \times I$)	262
Figure 7.5	Accuracy of $\nabla_\sigma J(\mu, \sigma)$ estimates on the 1-D function $f(x) = x^5$ ($\mu = 1$ and varying σ)	266
Figure 7.6	$\nabla_\sigma J(\mu, \sigma)$ accuracy on the 1-D function $f(x) = x^5$ ($\sigma = 0.1$ and varying μ)	268
Figure 7.7	uNES/NES comparison on parabolic function	271
Figure 7.8	uxNES/xNES comparison on a 10-D parabolic function	282
Figure 7.9	Estimate of $\nabla_\mu J(\theta)$ with 10-D parabolic function using fitness shaping	283
Figure 7.10	Average uxNES fitness progression on 40-D problems with different learning rates	288
Figure 7.11	Average uxNES $tr(C)$ progression on 40-D problems with different learning rates	289
Figure 7.12	Average xNES/uxNES/USA-ES fitness progression on test problems	292
Figure 7.13	uxNES global optimization algorithm results for 10-D test functions	296
Figure 7.14	uxNES results on a 30-node lunar lander problem	297
Figure C.1	Unscented RCGA 30-node lunar lander trajectory	356

THIS PAGE INTENTIONALLY LEFT BLANK

List of Tables

Table 2.1	RLV problem constants	34
Table 2.2	Listing of RLV state and control variables and ranges	38
Table 2.3	Initial/final conditions for RLV footprint problem	41
Table 3.1	A comparison of Hamming distances in binary and reflective binary gray encodings	67
Table 4.1	Attributes of parallel island GAs used for penalty experiments . .	128
Table 4.2	Method comparison GA attributes	147
Table 4.3	GA type comparison results	153
Table 5.1	Suggested Values For CMA-ES Coefficients	182
Table 5.2	xNES recommended parameters	192
Table 6.1	Multimodal objective functions for benchmark problem set	202
Table 6.2	10-D USA-ES vs. RSA-ES results ($\rho = 10 \lambda = 21$)	212
Table 6.3	20-D USA-ES vs. RSA-ES results ($\rho = 20 \lambda = 41$)	213
Table 6.4	40-D USA-ES vs. RSA-ES results ($\rho = 40 \lambda = 81$)	213
Table 6.5	USA-ES vs. RSA-ES success probabilities	216
Table 6.6	USA-ES/RSA-ES 10D solution quality results	217
Table 6.7	USA-ES/RSA-ES 20D solution quality results	218
Table 6.8	USA-ES/RSA-ES 40D solution quality results	218
Table 6.9	USA-ES average gradient L2 norms of final solutions	223
Table 6.10	CMA-ES user-defined strategy parameter settings	225

Table 6.11	10-D USA-ES vs. CMA-ES results ($\rho = 10 \lambda = 21$)	225
Table 6.12	20-D USA-ES vs. CMA-ES results ($\rho = 20 \lambda = 41$)	226
Table 6.13	40-D USA-ES vs. CMA-ES results ($\rho = 40 \lambda = 81$)	226
Table 6.14	USA-ES vs. CMA-ES success probabilities	228
Table 6.15	USA-ES/CMA-ES 10D solution quality results	229
Table 6.16	USA-ES/CMA-ES 20D solution quality results	229
Table 6.17	USA-ES/CMA-ES 40D solution quality results	229
Table 6.18	10-D USA-ES ($\lambda_{USA} = 21$) vs. GHSA-ES ($\lambda_{GHSA} = 221$)	233
Table 6.19	20-D USA-ES ($\lambda_{USA} = 41$) vs. GHSA-ES ($\lambda_{GHSA} = 841$)	234
Table 6.20	40-D USA-ES ($\lambda_{USA} = 81$) vs. GHSA-ES ($\lambda_{GHSA} = 3281$)	234
Table 6.21	USA-ES vs. GHSA-ES success probabilities	236
Table 6.22	USA-ES vs. RSA-ES parameters for the lunar lander problem . .	242
Table 6.23	USA-ES vs. RSA-ES solutions to the lunar lander problem	242
Table 6.24	USA-ES parameters for the lunar lander problem	244
Table 6.25	USA-ES solutions to the lunar lander problem	244
Table 7.1	Accuracy of $\nabla_\mu J(\mu, \sigma)$ on the 1-D functions $f(x) = x^2$ and $f(x) = x^9$ ($\mu = 0, \sigma = 1$)	255
Table 7.2	$\nabla_\mu J(\mu, \sigma)$ error norms on the 1-D function $f(x) = x^5$ ($\mu = 1$ and varying σ)	258
Table 7.3	$\nabla_\mu J(\mu, \sigma)$ relative error on the 1-D function $f(x) = x^5$ ($\mu = 1$ and varying σ)	259
Table 7.4	$\nabla_\mu J(\mu, \sigma)$ error norms on the 1-D function $f(x) = x^5$ ($\sigma = 0.1$ and varying μ)	260
Table 7.5	$\nabla_\mu J(\mu, \sigma)$ relative error on the 1-D function $f(x) = x^5$ ($\sigma = 0.1$ and varying μ)	261

Table 7.6	$\nabla_\mu J(\mu, \sigma)$ accuracy on 10-D function $f(\bar{x}) = \sum_{i=1}^n x_i^5$ ($\bar{\mu} = [1, \dots, n]^T$ and $C = 0.1 \times I$)	262
Table 7.7	$\nabla_\sigma J(\mu, \sigma)$ error norms on the 1-D function $f(x) = x^5$ ($\mu = 1$ and varying σ)	266
Table 7.8	$\nabla_\sigma J(\mu, \sigma)$ relative error on the 1-D function $f(x) = x^5$ ($\mu = 1$ and varying σ)	267
Table 7.9	$\nabla_\sigma J(\mu, \sigma)$ error norms on the 1-D function $f(x) = x^5$ ($\sigma = 0.1$ and varying μ)	268
Table 7.10	$\nabla_\sigma J(\mu, \sigma)$ relative error on the 1-D function $f(x) = x^5$ ($\sigma = 0.1$ and varying μ)	269
Table 7.11	uNES/NES comparison parameters	271
Table 7.12	uxNES/xNES comparison	283
Table 7.13	uxNES average best fitness vs. covariance learning rates	287
Table 7.14	10-D xNES/uxNES/USA-ES results ($\lambda = 21$)	290
Table 7.15	20-D xNES/uxNES/USA-ES results ($\lambda = 41$)	290
Table 7.16	40-D xNES/uxNES/USA-ES results ($\lambda = 81$)	291
Table 7.17	uxNES results on a 30-node lunar lander problem	297
Table A.1	Chapter 1 symbol definitions	315
Table A.2	Chapter 2 symbol definitions	316
Table A.3	Chapter 3 symbol definitions	317
Table A.4	Chapter 4 symbol definitions	320
Table A.5	Chapter 5 symbol definitions	322
Table A.6	Chapter 6 symbol definitions	325
Table A.7	Chapter 7 symbol definitions	326
Table A.8	Chapter 8 symbol definitions	327
Table A.9	Appendix B symbol definitions	328

Table A.10	Appendix C symbol definitions	330
Table C.1	Unscented RCGA parameters	355
Table C.2	Unscented RCGA lunar lander results	355

List of Acronyms and Abbreviations

3DOF	three degrees of freedom
ANOVA	analysis of variance
CMA-ES	covariance matrix adaptation evolution strategy
CPU	central processing unit
CSA-ES	cumulative self-adaptation evolution strategy
DGX	dynamic Gaussian crossover
DoD	Department of Defense
DTGX	dynamic truncated Gaussian crossover
EPP	Evolutionary Progress principle
ES	evolution strategy
GA	genetic algorithm
GHSA-ES	Gauss-Hermite simulated annealing evolution strategy
GPU	graphics processing unit
GR	genetic repair
ICBM	intercontinental ballistic missile
KKT	Karush-Kuhn-Tucker
MCMC	Markov Chain Monte Carlo
MIMD	multiple instruction multiple data
MPI	message passing interface

NES	natural evolution strategy
NLP	nonlinear program
NPS	Naval Postgraduate School
OCP	optimal control problem
RCGA	real-coded genetic algorithm
RLV	reusable launch vehicle
RSA-ES	random simulated annealing evolution strategy
SBX	simulated binary crossover
SIMD	single instruction multiple data
SPMD	single program multiple data
SPX	simplex crossover
UNDX	unimodal distribution crossover
uNES	unscented natural evolution strategy
USA-ES	unscented simulated annealing evolution strategy
USG	United States government
USN	U.S. Navy
uxNES	unscented exponential natural evolution strategy
xNES	exponential natural evolution strategy

Executive Summary

The overarching goal of this dissertation is to investigate evolutionary optimization techniques and improve them through the use of parallel computation and unscented sampling. Evolutionary algorithms are selected as the focus area of this research due to their compatibility with parallel computation coupled with other unique characteristics that prove to be beneficial when addressing several challenges faced by many gradient-based optimization algorithms. These characteristics include the potential to work in discontinuous spaces, the ability to work with non-differentiable search domains and the capability to address black box optimization problems. Evolutionary algorithms represent a subset of Markov Chain Monte Carlo techniques that include both genetic algorithms (GAs) and evolution strategies (ESs), where stochastic processes are utilized to explore search domains. This dissertation begins with a discussion of relevant, real-world applications of optimization and optimal control within astronautical engineering. In conjunction with this, a number of numerical optimization and optimal control techniques are reviewed. The key challenges associated with evolutionary algorithms are identified and translated into three overarching goals to be addressed by this dissertation. The first goal is to investigate and employ techniques that enable evolutionary algorithms to effectively handle constraints in a way that allows for feasible solutions to constrained optimization problems. The second goal is to improve computation times and efficiencies associated with evolutionary algorithms. The last goal is to enhance the evolutionary algorithm's robustness and ability to consistently find accurate solutions within a finite number of iterations. With these overarching goals defined, the remainder of this work focuses on addressing them through innovative modifications to various forms of GAs and ESs. The following list offers the reader a high-level summary of the notable contributions and innovations that have stemmed from this research.

Parallel Reusable Launch Vehicle (RLV) Landing Footprint Determination

An application of parallel computation is used to rapidly generate RLV landing footprints. This study is included in this dissertation to illustrate the potential benefits associated with parallel computation, a technique that is used throughout this dissertation. While the parallel computing aspects of this study are not necessarily novel or unique, this specific application has not been documented in literature or fully explored to this extent. Further-

more, the insights of appropriately balancing computational workload coupled with parallel processing memory architectures are useful in the context of this real-world engineering optimal control problem and are leveraged throughout this dissertation. It must be noted that workload balancing and memory architecture studies have been done extensively, however, it is unique to study them through the lens of an RLV optimal control engineering application. This work has been presented and published for the 2015 American Astronautical Society (AAS)/American Institute of Aeronautics and Astronautics (AIAA) Astrodynamics Specialist Conference in Vail, Colorado [1].

Parallel GA Exact Penalty Function Analysis

To address the first goal of effective constraint handling, a unique set of trade studies are used to explore the performance of exact penalty functions within the context of parallel GAs. It is discovered that the exact penalty functions, though not as frequently applied in GAs, seem to produce better results when compared to inexact forms of penalty functions. This is worthwhile to note, given that evolutionary algorithms do not require continuously differentiable functions and therefore are able to work with exact penalty functions.

Parallel GA Type and Optimal Control Trade Study

In order to address all three goals as they relate to GAs, six different parallel GA types are studied as they apply to a lunar lander optimal control problem using exact penalty methods. Trades are conducted on parameters that adapt the parallel architecture of these various GA types. It is shown that parallel processing not only improves accuracy and convergence speed, but also significantly improves algorithm robustness by reducing the algorithm sensitivity to optimally tuned user-defined parameters. This work has been presented and published for the 2016 AAS/AIAA Spaceflight Mechanics Conference in Napa, California [2].

Dynamic Gaussian-Based Real-Coded GA (RCGA) Crossover Operators

In an attempt to further improve algorithm accuracy and efficiency, two new Gaussian-based dynamic covariance crossover operators are introduced. The dynamic truncated Gaussian crossover (DTGX) operator and dynamic Gaussian crossover (DGX) operator are developed and investigated as part of a unique study of parallel RCGAs as they apply to optimal control [2]. These new operators are created to introduce several characteristics

that are not typically found in RCGA crossover operators. More specifically, they utilize Gaussian distributions in a way that favors solutions with better objective function values and have the ability to vary the search resolution by changing the distribution covariance over time. These new crossover operators achieve similar performance when compared to state-of-the-art RCGA crossover operators. This exercise proves to be valuable given that it provides useful insights and lessons that help lead to the introduction of an unscented RCGA.

Unscented Sampling In RCGAs

A new form of RCGA is introduced that utilizes deterministic sampling within a crossover operator. This unscented RCGA proves to be a critical stepping stone for the development of the various ESs that utilize unscented sampling techniques. It is empirically demonstrated that this new form of unscented RCGA is able to achieve very accurate results (within $1.9 \times 10^{-3}\%$ of the known optimal solution) in solving a lunar lander optimal control problem when run on multiple processors in a parallel island architecture. This technique is shown to help address the goals of improving computational speed and solution accuracy. It will be seen that this idea of an unscented RCGA is still very much in its infancy, and several new ideas are proposed in the last chapter that may further improve this algorithm.

Unscented Sampling In ESs

ESs prove to be a more natural fit for the application of unscented sampling when compared to standard RCGAs. With this, a transition is made and the idea of unscented sampling is applied to the construct of an ES. Initially, a fairly simple ES is developed by building upon the idea of the DGX crossover operator that was introduced in Chapter 4. The unscented simulated annealing evolution strategy (USA-ES) and Gauss-Hermite simulated annealing evolution strategy (GHSA-ES) are both created by applying a DGX-inspired mutation operator that uses unscented sampling within the general construct of an ES. Empirical analysis shows that this new algorithm obtains enhanced performance in terms of accuracy and computational time when compared with a similar ES that utilizes Monte Carlo sampling. Both third-order unscented sigma points and higher order sparse Gauss-Hermite points are used within this algorithm construct. These new unscented ESs are able to consistently find globally optimal solutions on the majority of the highly multimodal benchmark test problems, even as the problem dimension is increased. These new ESs that

use deterministically chosen points are shown to outperform the state-of-the-art covariance matrix adaptation evolution strategy (CMA-ES) by consistently finding globally optimal solutions with lower numbers of required function evaluations. It must also be noted that a parallel island implementation of this new USA-ES is developed and applied to the lunar lander problem as another unique contribution and proof of concept. All of these results indicate that this contribution may prove to be a very useful in addressing all of the goals regarding evolutionary algorithms. A portion of this work has been presented and published for the 2016 AAS/AIAA Spaceflight Mechanics Conference in Napa, California [3].

Unscented Sampling In The Natural Evolution Strategy (NES)

The sophistication and performance of utilizing unscented sampling within the context of an ES is further increased by integrating this idea into the state-of-the-art NES. This allows for the full utilization of the unscented abscissas where both the nodes and the associated weights are used to evolve both the distribution mean, and the distribution covariance. This is a major, unique contribution that has shown great promise in solving the challenging set of benchmark problems from literature. Empirical results indicate that this new unscented exponential natural evolution strategy (uxNES) is able to find globally optimal solutions for many of the multimodal problems where many of the other methods fail. Lastly, this new uxNES produces near-optimal, feasible results for a lunar lander problem much faster than other evolutionary methods investigated without the use of parallel processing. A summary paper of this work has been submitted for publication and will be presented at the AIAA Space 2016 Conference in Long Beach, CA [4].

uxNES Global Optimization Algorithm

To further address the last goal of consistently finding sufficiently accurate solutions within a finite number of iterations, this new uxNES algorithm is integrated into a global optimization scheme using the concepts of a reigning optimal and fitness scaling to further enhance the capabilities of the uxNES. The method quickly eliminates locally optimal points that are equivalent to, or worse than, the reigning optimal solution through the use of exact penalty techniques. The new global optimizer proves to enhance solution quality on the more challenging problems considered in this study. This is a unique contribution in and of itself that produces promising empirical results warranting further research and development.

In addition to the listed contributions several literature reviews and concept overviews are presented throughout this dissertation. Prior to applying parallel computation to the various algorithms in this work, the various techniques and associated computational hardware typically used for parallel computing are reviewed. In addition, an extensive literature review of GAs is presented before the unique studies and innovations regarding GAs are explored. The standard binary-encoded GAs along with RCGAs are developed and presented in great detail. A number of theoretical aspects to include various advantages and disadvantages associated with the various types of GAs are discussed at length. Current state-of-the-art techniques associated with several different GA types are also covered. In addition, this literature review investigates various methods and program architectures used for running these types of algorithms in parallel. All of this information is utilized as a foundation that is then built upon to investigate new areas and develop new techniques associated with GAs.

As the dissertation transitions from GAs to ESs, an in-depth literature review on ESs is presented. This review further investigates the standard techniques and operations commonly associated with ESs. It discusses advantages, drawbacks, and theoretical aspects associated with various forms of these optimization algorithms. In addition, the various state-of-the-art ESs are presented and discussed in detail. Again, this survey and summary of findings provides the necessary information that is then built upon to produce a series of unique contributions that improve upon the state-of-the-art in ESs.

Lastly, this dissertation concludes with a discussion of several relevant astronautical engineering applications that the newly developed evolutionary algorithms are particularly well-suited for. In addition, a number of future work ideas and proposed improvements to the unscented RCGA and the uxNES are introduced and discussed. The major contributions of this work are shown to directly address the overarching challenges that many evolutionary algorithms face. The enhanced algorithms developed in this dissertation are shown to consistently outperform a number of state-of-the-art evolutionary algorithms in terms of efficiency (number of function evaluations) and solution accuracy (difference in objective function value from the known globally optimal solution) on many difficult optimization problems from literature. In addition, they produce promising results when applied to a lunar lander optimal control problem, indicating their potential usefulness on a number of other real-world astronautical application problems.

List of References

- [1] C. McGrath, M. Karpenko, and R. J. Proulx, “Distributed computation for near real-time footprint generation,” in *Proceedings of The 2015 AAS/AIAA Astrodynamics Specialist Conference*, Vail, CO, 2015.
- [2] C. McGrath, M. Karpenko, and R. J. Proulx, “Parallel genetic algorithms for optimal control,” in *Proceedings of The 2016 AAS/AIAA Spaceflight Mechanics Conference*, Napa, CA, 2016.
- [3] C. McGrath, M. Karpenko, R. J. Proulx, and I. M. Ross, “Unscented evolution strategies for solving trajectory optimization problems,” in *Proceedings of The 2016 AAS/AIAA Spaceflight Mechanics Conference*, Napa, CA, 2016.
- [4] C. McGrath, M. Karpenko, R. J. Proulx, and I. M. Ross, “An unscented natural evolution strategy for solving trajectory optimization problems,” in *Proceedings of The AIAA Space 2016 Conference*, Manuscript Submitted. Long Beach, CA, 2016.

Acknowledgments

I first need to thank my family for all of their love and support along the way. For me, this work along with life itself would not be possible without them. Next, this work has come to fruition by me standing on the shoulders of intellectual giants. It has been an honor and privilege to work with some of the best minds in academia. My advisor, Dr. I. Michael Ross, has been an amazing source of creativity and outside-the-box thinking throughout this process. His ability to explain some of the most complex concepts in an eloquent and succinct manner has been invaluable. Dr. Ronald Proulx never ceases to amaze me with his unending supply of innovative solutions to even the most challenging of problems. This, coupled with his ability to troubleshoot and quickly identify potential issues, has proven to be crucial throughout this work. Dr. Mark Karpenko has also been a critical element to this research process with his keen eye for detail and his extraordinary ability to see the big picture. Like Dr. Proulx, he also has been instrumental in establishing new ideas and innovative techniques relative to the algorithms in this work. I would also like to thank Dr. Wei Kang, Dr. Lucas Wilcox, and Dr. Isaac Kaminer for being very helpful throughout this process. Without their knowledge and instruction on a number of topics central to this dissertation, none of this would have been possible.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1:

Optimization and Optimal Control

1.1 Introduction

Optimization and optimal control have become critical enablers to many astronautical engineering processes and systems, as discussed throughout this chapter. This dissertation investigates and develops several new ideas and techniques associated with evolutionary computation that have the potential to improve the current state-of-the-art in solving various types of optimization and optimal control problems. This research is conducted to further understand and improve several of these evolutionary algorithms that may play a vital role in improving current space systems and developing new ones. The focus of this work is to improve upon the state-of-the-art genetic algorithm (GA) and evolution strategy (ES) by addressing several overarching challenges associated with these types of algorithms. The application of various forms of exact penalty methods coupled with parallel computation and unscented sampling techniques are shown to enhance the performance of several forms of evolutionary algorithms. More specifically, GAs and ESs are modified and enhanced to directly address challenges associated with constraint handling, computational speed, algorithm efficiency, and the ability to consistently find accurate solutions within a finite number of iterations. These new techniques are shown to improve upon the state-of-the-art evolutionary algorithms when applied to a set of challenging multimodal optimization problems from literature. In addition, these new methods produce promising solutions to a lunar lander optimal control problem indicating their usefulness and applicability to a historically difficult class of problems.

This chapter begins by discussing some of the relevant applications of optimization and optimal control in real-world space systems. Next, some of the numerical optimal control and more general optimization techniques are introduced. From here, the chapter highlights some of the challenges associated with many mainstream optimization and optimal control techniques. It then provides the reader with a brief introduction to the No Free Lunch theorem as it applies to optimization algorithms. Next, a summarized list of unique contributions is presented. Lastly, the chapter concludes with a top level summary of the

chronological research path taken outlining the layout of the content as it is organized in the remainder of this document.

1.2 Several Optimization Problems in Astrodynamics

There is a wide range of applications for optimization techniques spanning from developing market forecasts in economics [1] to automatically composing jazz solos in music [2]. However, this dissertation and this section focus on a few of the many applications specific to astronautical engineering. As academia and industry alike continue down the road of improving efficiencies and advancing capabilities of existing and future space systems, optimization and optimal control become increasingly important. This section introduces several important applications of optimization and optimal control that continue to be active areas of research within the space launch, satellite, and missile system communities.

1.2.1 Space Launch

There are many interconnected systems and design problems that can be optimized within the realm of space launch. The optimization of launch trajectories is a challenging optimal control problem that has been studied since the beginning of space launch [3]. Trajectory optimization can be difficult due to the dynamic conditions encountered in the various phases of launch due to staging, changing weather, dynamic atmospheric conditions, and variances between expected and actual system performance [4]. While the optimization of standard launch trajectories can be challenging, future launch system concepts are even more complex due to the addition of partial or full re-usability. This requires optimal ascent and return trajectories where there are tight tolerances on reentry due to aerodynamic heating limitations coupled with design limitations on thermal protection systems [5]. The growing interest in reusable launch systems will continue to push the need for increasingly sophisticated optimization techniques that can enhance efficiencies and system performances under dynamic and uncertain conditions.

The optimal design of launch systems in terms of capability and efficiency is another problem that continues to be an active area of study [6]–[8]. This process requires the optimization of a large number of interconnected subsystems that all need to produce outputs that lead to a global system that can be optimized to a set of top-level metrics. The optimal design point

for any one of the subsystems may not necessarily be the optimal design point that leads to the best overall performance of the higher-level system. To add even more complexity to this task, the launch systems are expected to perform over a range of payload sizes, weights, and delivery orbits. This again results in a problem where the entire trade space of potential solutions needs to be considered and the best solution for any given mission will likely not be the best global solution to optimize efficiency and performance across all of the mission requirements [9]. It can likely be inferred that trajectory optimization, while a challenging optimization problem in and of itself, is just a small part of a larger problem. Any error, or inefficiencies in the optimal trajectory determination can ripple through the entire system design and can introduce even more inefficiency throughout the system. This of course is also the case with the optimization of many other subsystems and components that make up the launch system.

Sustaining and improving current launch systems is another area in space launch that can likely benefit from the application of optimization techniques. By improving the optimality of current launch trajectories, a number of derived benefits can potentially be obtained. As an example, real-time optimal control could potentially increase system efficiencies and confidence levels so that lower safety margins could be tolerated while still meeting the systems expected reliability requirements. This could be anything from decreasing the excess propellant requirements to alleviating strict launch window weather requirements which could further improve launch flexibility [4]. This idea of enabling the reduction of design margins on existing systems due to optimization and optimal control could be applied to a number of subsystems throughout the launch system, potentially reducing cost and time associated with meeting tight design requirements and tolerances on various components.

1.2.2 Satellites and Space Vehicles

There are also numerous applications for optimal control and optimization on satellite systems and space vehicles. Space mission design for deep-space activities is another area of research that requires the solution of hybrid optimal control problems to find an optimal mission design trajectory [10], [11]. A hybrid optimal control problem is one that has both continuous and discrete decision variables. For a deep space mission, the discrete decision variables can describe whether or not the space vehicle will perform flybys of certain planets or asteroids to obtain gravity assists. The continuous variables can describe

the trajectories and arcs between the various flyby maneuvers. In addition, the departure and arrival times are continuous variables that play a major role in how the other planetary objects and asteroids line up for a given portion of the trajectory. These problems can be challenging due to the fact that they deal with both continuous and discrete variables where combinations of optimization and optimal control techniques are often required to generate near-optimal solutions.

The design of satellite constellations is another important application of optimization pertaining to satellites. A great deal of research and effort has gone into designing satellite constellations for various mission requirements [12], [13]. Given the expense required to develop, build, test, and launch satellites, designing a constellation that allows for all of the mission requirements to be met with a minimal number of satellites is a very critical task. Constellation design is another problem that can consist of both continuous and discrete decision variables. The number of satellites per orbit, and the number of different orbits used for the constellation are discrete variables, while the orbital elements that describe the properties of each orbit are continuous. Constellation design is a task that can greatly influence the early design phases of a satellite system and can lead to significant cost savings and efficiencies. However, like launch vehicle design, the satellite system design process has to consider a number of factors to include aspects of the system such as: available ground stations, downlink/uplink rates, sensor capabilities, battery life, eclipse times, thermal properties, etc. [14]. Inputs from the entire design team are often needed to help define requirements and identify trades to be optimized. In addition, mission requirements are not always black and white, and several different missions may be desired from a single system. In these scenarios, there are likely to be multiple objectives that need to be considered simultaneously.

Given the limited amount of on-board resources, simple satellite maneuvers, and station-keeping operations can be critical to the point that even the smallest amount of saved propellant or reduction in time can be very important. While the satellites are still in the design phase, optimization techniques can be used to help determine the placement and type of attitude control components on a satellite. This can lead to better satellite design configurations for the various required maneuvers and station-keeping operations expected on a given mission. In addition to this, a number of examples can be found where optimal control and optimization techniques have been applied to develop minimum effort

or minimum time maneuvers for existing satellites that have already been designed and launched [15], [16]. One great example of this is where the International Space Station was able to completely rely on control moment gyros and gravity gradients to perform a maneuver that had previously been done using thrusters and valuable propellant [15]. This was made possible by an innovative application of optimal control that resulted in significant cost savings for NASA [15]. There have also been interesting examples of optimal control and optimization techniques being applied to satellites with degraded capabilities (e.g. loss of a reaction wheel) to retain full mission capability by utilizing other on-board systems, or forces such as solar pressure and gravity gradients [17], [18]. Lastly, there are a number of oversubscribed satellite systems that require optimal task scheduling to maximize the amount of science data, commercial images, etc. collected in a way that optimizes the use of the system [19]–[22]. This idea is further discussed and developed in Chapter 8 as some of the techniques developed in this dissertation are potentially well-suited for these types of applications.

A number of the problems discussed in this chapter can be formulated in a way that requires both discrete and continuous decision variables, resulting in hybrid optimization problems. As an example, the scheduling of collections for a task-saturated commercial imaging satellite could be formulated as a hybrid optimization problem that requires both types of decision variables. Figure 1.1 illustrates a simple version of this problem where the green circles represent desired targets. It can be seen that the field of view of the camera on the satellite can trace out any number of ground tracks subject to the constraints of the satellite system, camera pointing requirements, orbital properties, etc. For illustration purposes, the red and black lines depict two potential paths that the satellite’s camera can take to collect images of two different sets of targets. This can be imagined as a traveling salesman problem with moving targets, where the route taken to cover a number of targets require the solution to an optimal control problem. Discrete decision variables are needed to determine whether or not a satellite takes a picture of a given target. These discrete decision variables not only define a single collect, but an entire ordered list of collection targets over a set time horizon. In addition to this, the trajectory that the satellite takes in terms of a slew maneuver to take pictures of the desired targets requires the solution of an optimization problem consisting of a number of continuous decision variables and constraints. It must be noted that solving hybrid optimal control problems is not the standard method currently

being used to solve satellite scheduling problems in industry, but it could be utilized in the future with the development of appropriate optimization algorithms.

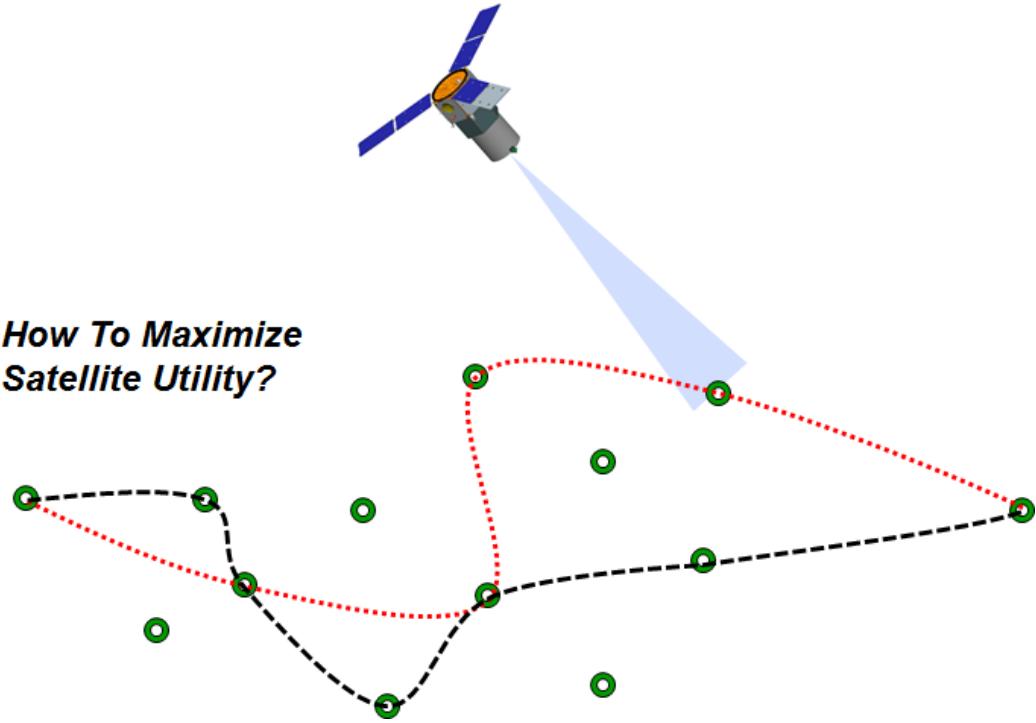


Figure 1.1. An oversubscribed commercial imaging satellite problem

Figure 1.2 illustrates one approach to solving this type of hybrid optimization problem where a combination of evolutionary algorithms and gradient-based solvers are utilized to find good solutions to these types of problems [10], [23]. It can be seen in Figure 1.2 that an evolutionary algorithm is applied to an outer-loop and is used to develop a discrete set of mission objectives that meet a set of user-defined constraints, given its ability to handle discrete variables. For each proposed set of mission objectives that the evolutionary algorithm finds, a gradient based solver is used to determine a trajectory that meets these objectives. This inner loop may also find that there are no possible trajectory solutions that can satisfy the proposed set of mission objectives. The evolutionary algorithm can then be used to propose a new set of mission objectives based on the previous results. This approach has been applied for several different hybrid optimization problems including deep space mission planning with multiple gravity assists [10], [11]. From this point, the inner/outer loop technique continues to iterate until a stopping criteria is met. The gradient based

solvers and evolutionary algorithms are further developed later in this chapter.

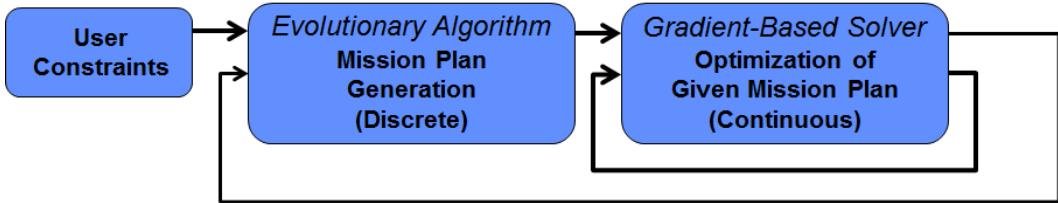


Figure 1.2. A multi-loop hybrid optimal control approach

1.2.3 Missile and Missile Defense

Optimization and optimal control are critical for the effectiveness of a number of current and future missile systems. intercontinental ballistic missiles (ICBMs) face some similar challenges to reusable launch vehicles (RLVs) such as needing to optimize trajectories with dynamic weather and atmospheric conditions while still accounting for thermal stresses associated with the descent phase. The location and strategic placement of permanent missile sites as well as missile defense sites can also be an important optimization problem. End-game maneuverability and additional range are critical for surface-to-air and air-to-air missile effectiveness [24]. Optimizing various approach trajectories in response to the target's evasive maneuvers can significantly affect a given missile system's capability and effectiveness [25]. This creates a very challenging optimal control problem that must be solved in real-time to adapt to dynamic conditions. Missile defense systems face similar challenges with even faster moving targets and an increased importance on optimal selection of missile site locations [26].

Another example where optimal control is needed stems from future missile systems being pursued as part of the prompt global strike initiative to deliver conventional weapons anywhere on the Earth within one hour [27]. A unique hypersonic glide trajectory is desired for these systems for several reasons: it removes potential ambiguity from heritage nuclear weapon delivery systems, it provides for more flexibility in avoiding overflight of third-party countries, and it can lead to increased accuracy [27]. Many of the proposed hypersonic glide delivery systems have challenging thermal environments that require the use of sophisticated thermal protection systems in conjunction with advanced optimal control techniques to keep temperatures and thermal gradients within tolerable ranges [28], [29].

1.3 Optimal Control Techniques and Associated Challenges

Optimal control problems can be thought of as a unique subset of optimization problems that have an infinite number of variables [30]. The standard form of an optimal control problem is depicted in Equation (1.1).

$$\begin{aligned} \bar{x}^T &\in \mathbb{R}^{N_x \times n} \quad \bar{u} \in \mathbb{R}^{N_u \times n} \\ \left\{ \begin{array}{l} \text{Minimize: } J[\bar{x}(\cdot), \bar{u}(\cdot), t_f] = E(\bar{x}_f, t_f) + \int_{t_0}^{t_f} F(\bar{x}(t), \bar{u}(t), t) dt \\ \text{Subject To: } \dot{\bar{x}} = f(\bar{x}, \bar{u}, t) \\ h(\bar{x}, \bar{u}) \leq \mathbf{0} \\ (t_0 - t(0), \bar{x}_0 - \bar{x}(0)) = (0, \mathbf{0}) \\ e(x_f, t_f) = \mathbf{0} \end{array} \right. \end{aligned} \quad (1.1)$$

In this form there are a set of state variables $\bar{x}(t, u) \in \mathbb{R}^{N_x \times n}$ that are necessary to describe the characteristics of the system of interest. Next, the control inputs that can be manipulated to generate a desired response are represented by $\bar{u}(t) \in \mathbb{R}^{N_u \times n}$. The state vector and control vectors have N_x and N_u number of continuous variables respectively. It is important to note that all optimal control problems must have states that are dependent on at least one control input (directly or indirectly) so that the system states can be changed in an optimal way to achieve a desired result. [30] The relationship between the states, controls, and time need to be captured in the dynamics functions that describe the changing conditions of the model $\dot{\bar{x}} = \bar{f}(\bar{x}, \bar{u})$. Once the state and control variables along with the state dynamics have been identified, the designer must develop a cost function that can appropriately capture the cost or characteristic of the problem that is to be minimized. It can be observed that the Bolza form of a cost function $J[\bar{x}(\cdot), \bar{u}(\cdot), t_f]$ is made up of an Mayer cost component, $E(\bar{x}_f, t_f)$, and a Lagrange cost component $\int_{t_0}^{t_f} F(\bar{x}(t), \bar{u}(t), t) dt$. [31] While this form indicates a minimization problem, any maximization problem can be converted into a minimization problem through a simple sign change. Optimal control problems are often subject to any number of initial conditions, \bar{x}_0 , and endpoint conditions $e(x_f, t_f)$. Path constraints, $h(\bar{x}, \bar{u})$,

are often added to enforce various constraints on state and control variables throughout the horizon of the problem. Lastly, optimal control problems can take various forms where the final time, t_f , the initial time, t_0 , the initial state, \bar{x}_0 , and the final state, \bar{x}_f , may be free variables, or they may be defined specifically by the problem formulation. All of this is put together in Equation (1.1) which illustrates the Bolza form of a continuous time optimal control problem [31].

There are several different techniques that can be used to solve these types of problems, each with their own unique advantages and disadvantages. The various methods that have been developed to solve optimal control problems are often categorized as either indirect or direct methods [32]. The primary difference between direct and indirect methods is that indirect methods utilize Pontryagin's principle, associated necessary conditions, and costate variables whereas the direct methods do not [33]. With this, both the shooting methods and collocation methods can be implemented either directly, or indirectly [32]. Indirect methods are sometimes considered to be more sensitive to an initial guess or starting point when compared to the direct methods, and are therefore not as common in application [30]. For this reason, only the direct versions of each numerical optimal control technique are discussed in this section.

1.3.1 Direct Shooting

Shooting methods are fairly popular for both launch vehicle trajectory optimization and orbit transfer problems where commercial codes such as Martin Marietta's POST have been developed for use on these types of problems [30]. These numerical methods utilize a discretization in time so that each continuous variable is determined over a set number of discrete time nodes, N_t . The shooting method utilizes a proposed control trajectory and any missing initial (or final) conditions to treat the problem as an initial (or final) value problem. At this point, a numerical time integration method is utilized to propagate the dynamics forward (or backward) in time to evaluate how well the proposed control trajectory performs in terms of cost and satisfying all of the constraints. It must be noted that the conditions for optimality (i.e. Pontryagin's principle) are not verified using this direct approach. This technique is illustrated in Equation (1.2) where the state can be determined at each time node based on a given control trajectory, \bar{u} , and initial conditions, \bar{x}_0 . In this specific case, a Euler numerical time integrator is utilized to integrate the ordinary differential equations,

$\dot{\bar{x}} = f(\bar{x}, \bar{u}, t)$, and approximate the states at each of the N_t discretized points. With the direct shooting approach, the state variables are solved for in a sequential fashion, as illustrated in Equation (1.2) [32]. In addition to solving for the state variables, all of the relevant problem constraints are enforced at each discretized point of the state and control trajectories.

$$\begin{aligned} \bar{x}^T &\in \mathbb{R}^{N_x \times N_t} \quad \bar{u} \in \mathbb{R}^{N_u \times N_t} \\ \bar{x} &\approx \begin{bmatrix} \bar{x}_1 = \bar{x}_0 + (t_1 - t_0) f(\bar{x}_0, \bar{u}_0, t_0) \\ \vdots \\ \bar{x}_{N_t-1} = \bar{x}_{N_t-2} + (t_{N_t-1} - t_{N_t-2}) f(\bar{x}_{N_t-2}, \bar{u}_{N_t-2}, t_{N_t-2}) \end{bmatrix} \end{aligned} \quad (1.2)$$

In addition, various quadrature and numerical integration methods can be used to solve for the Mayer term in the cost function as depicted in Equation (1.3) below.

$$J[\bar{x}(\cdot), \bar{u}(\cdot), t_f] = \int_{t_0}^{t_f} F(\bar{x}(t), \bar{u}(t), t) dt \approx \sum_{k=0}^{N_t-1} (t_{k+1} - t_k) F(\bar{x}_k, \bar{u}_k, t_k) \quad (1.3)$$

This method can effectively convert the continuous time optimal control problem into a nonlinear program (NLP) through the discretization of time and the use of a numerical integration technique. After the problem has been formulated in this way, one of many NLP solvers may be utilized to solve for an optimal control trajectory that meets all of the requirements. A generic example of a simple direct shooting method is outlined in Equation (1.2). It can be seen that the optimal control problem can be converted into a NLP through this process. One of the primary disadvantages with shooting methods is that small errors can quickly become large as they are numerically propagated through time [32]. This issue can become increasingly significant as the time span of a given problem grows. Multiple shooting methods have also been developed to help remedy this issue by splitting the shooting problem up into multiple sections. With these multiple shooting techniques, the issue of error propagation is not entirely avoided and more decision variables are required for the same problem when compared to a single shooting approach. [30]

1.3.2 Direct Collocation

The direct collocation, or transcription, method utilizes a numerical differentiation technique, such as the simple finite difference approximation outlined in Equation (1.4), to define

the dynamics at each discrete time node as an equality constraint that must be satisfied. The last line in Equation (1.4) becomes the equality constraint that must be enforced at each time node for all of the states. In addition, all of the other constraints, endpoint conditions, and initial conditions must be enforced at each discrete point in time. This method is not as vulnerable to the error propagation issue that plagues shooting methods due to the fact that all variables are solved for simultaneously, however it does create a larger number of NLP decision variables that must be solved. It also results in a large number of equality constraints that must be accounted for. In collocation methods, the control variables and the state variables at each discretized point in time must be solved for. While this can result in very large problems, when compared to the shooting methods, it does avoid some of the issues associated with error propagation. With collocation, an initial guess for all of these variables is also needed, as will be discussed in Section 1.3.3. It must be noted that advances in computing technologies coupled with techniques to leverage sparse matrix characteristics have significantly helped alleviate this issue of large problem sizes and have made collocation techniques increasingly practical in application [30].

$$\begin{aligned}\dot{\bar{x}}_k &= f(\bar{x}_k, \bar{u}_k, t_k) \\ \dot{\bar{x}}_k &\approx \frac{\bar{x}_{k+1} - \bar{x}_k}{t_{k+1} - t_k} \\ \frac{\bar{x}_{k+1} - \bar{x}_k}{t_{k+1} - t_k} &\approx f(\bar{x}_k, \bar{u}_k, t_k) \\ \bar{x}_{k+1} - [(t_{k+1} - t_k) f(\bar{x}_k, \bar{u}_k, t_k) + \bar{x}_k] &\approx \mathbf{0}\end{aligned}\tag{1.4}$$

By enforcing these as constraints, a new NLP representing the discretized optimal control problem is illustrated in Equation (1.5). In direct collocation, the state variables and the control variables all become decision variables that must be solved for. Where, the direct shooting method is setup so that only the control variables and any missing initial/final conditions become the decision variables. When using Euler discretization, direct shooting and direct collocation methods can look very similar, but the ideas behind each approach are fundamentally different. The primary difference between the two methods when using Euler's method is that a direct shooting method solves for the state variables sequentially, while a direct collocation technique solves for all of the state variables and control variables simultaneously [32].

$$\bar{x}^T \in \mathbb{R}^{N_x \times N_t} \quad \bar{u} \in \mathbb{R}^{N_u \times N_t}$$

$$\left\{ \begin{array}{l} \text{Minimize: } J[\bar{x}(\cdot), \bar{u}(\cdot), t_f] = \int_{t_0}^{t_f} F(\bar{x}(t), \bar{u}(t), t) dt \approx \sum_{k=0}^{N_t-1} (t_{k+1} - t_k) F(\bar{x}_k, \bar{u}_k, t_k) \\ \text{Subject To: } \bar{x}_1 - [(t_1 - t_0) f(\bar{x}_0, \bar{u}_0, t_0) + \bar{x}_0] = \mathbf{0} \\ \vdots \\ \bar{x}_{N_t-1} - [(t_{N_t-1} - t_{N_t-2}) f(\bar{x}_{N_t-2}, \bar{u}_{N_t-2}, t_{N_t-2}) + \bar{x}_{N_t-2}] = \mathbf{0} \\ (t_0 - t(0), \bar{x}_0 - \bar{x}(0)) = (0, \mathbf{0}) \\ e(\bar{x}_{N_t-1}, t_{N_t-1}) = \mathbf{0} \end{array} \right. \quad (1.5)$$

While a simple Euler collocation method is illustrated in this example, more accurate methods are often used, such as the Hermite-Simpson method [30]. Again, it can be seen that this collocation technique will eventually result in a large NLP after the dynamic equations have been converted into large sets of equality constraints. Given that both techniques for numerically solving optimal control problems result in NLPs that must be solved, it is important to briefly introduce techniques used to solve NLPs and the associated challenges with these methods.

1.3.3 NLP Techniques

The standard form of an NLP with both inequality constraints, $g_i(\bar{x}) \leq 0$, and equality constraints, $h_k(\bar{x}) = 0$, is depicted in Equation (1.6) where l is the number of inequality constraints, m is the number of equality constraints, and $\bar{x} \in \mathbb{R}^n$ is a n -dimensional solution vector. The search domain \mathbb{R}^n is often defined by upper and lower bounds on the state variables $\bar{x}^{UB} \leq \bar{x} \leq \bar{x}^{LB}$. The feasible set \mathbb{F} is a subspace of \mathbb{R}^n that is created by the intersection of all constraints and state bounds ($g_i(\bar{x}) \cup h_k(\bar{x}) \quad \forall i, k \quad s.t. \quad \bar{x} \in \mathbb{R}^n$). Direct techniques used for optimal control ultimately translate the continuous time optimal control problems into large NLPs of this form. Given that this standard formulation is used to define a large number of optimization problems, various techniques for solving and optimizing these types of problems are the primary focus for much of this research.

$$\begin{aligned}
& \text{minimize } f(\bar{x}) \\
& \text{subject to:} \\
& g_i(\bar{x}) \leq 0 \quad i = 1, \dots, l \\
& h_k(\bar{x}) = 0 \quad k = 1, \dots, m
\end{aligned} \tag{1.6}$$

There have been a number of techniques used for solving NLPs. For unconstrained NLPs, methods such as gradient search, Newton's method, Quasi-Newton methods, and BFGS search can be implemented [34]. For constrained NLPs, additional methods such as Lagrange Multiplier techniques, barrier/penalty methods, reduced gradient algorithms, quadratic programming methods, separable programming methods, and polynomial geometric programming techniques can be applied [34]. Many of these techniques are gradient-based methods that utilize gradient and Hessian information to guide iterative algorithms toward optimal points in the search space. Gradient-based methods can converge very quickly and produce very accurate results when compared to stochastic methods and evolutionary algorithms [33]. The capabilities of many of these gradient-based methods in terms of feasible problem sizes and computational speed have been improving along with the growth of computational capabilities. As an example, in the 1970s and 1980s, typical problem sizes had dimensions that were under 100, however by the 1990s it was not uncommon to see problem dimensions that were upwards of 100,000 [30]. Problem size limits have been even further increased through leveraging sparse matrix properties in codes such as SNOPT [30].

Unfortunately, many of these gradient-based methods share a couple of overarching shortfalls. First of all, many of these methods require continuously differentiable search domains, making it difficult to deal with discrete decision variables and exact penalty functions [33]. Next, gradient-based algorithms are fairly sensitive to the selected starting point or initial guess. Conventional, gradient based techniques often find locally optimal solutions that are in the neighborhood of the initial starting point, making it difficult to consistently find globally optimal solutions on problems that are multimodal [33].

1.3.4 Evolutionary Algorithms

There are a number of evolutionary algorithms that all depend on stochastic-based search processes which can also be applied to solve unconstrained optimization problems. With the addition of appropriate constraint handling techniques, these algorithms can be used to solve NLPs as well [33]. These evolutionary optimization algorithms typically fall under the larger umbrella of Markov Chain Monte Carlo (MCMC) methods that share the common characteristic of forming various forms of Markov Chains to progress from an initial state to a final state [35]. In fact, many iterative computer simulations and algorithms fit into this broad category of MCMC as Geyer states:

Thus most simulations can be thought of as MCMC if the entire state of the computer program is considered the state of the Markov chain. Hence, MCMC is a very general simulation methodology. [35]

Markov chains take the form X_1, X_2, \dots, X_n where X represents a state and the probability of transitioning from state X_n to state X_{n+1} is only dependent on state X_n [35]. Typically, the term evolutionary algorithm refers to GAs and ES [36]–[38]. However, algorithms such as Particle Swarm Optimization, Ant Colony Optimization, and Differential Evolution share similar traits and have been grouped into this category of evolutionary algorithms on occasion [33]. This dissertation primarily focuses on the development and application of new techniques within the constructs of GAs and ESs. GAs and ESs are both algorithms that are loosely based on evolutionary processes that naturally occur in biology. They were originally introduced in the mid 1960s to early 1970s [39], [40]. It must be noted that full literary reviews and much more in-depth descriptions of both GAs and ESs are presented in Chapter 3 and Chapter 5 respectively.

Evolutionary algorithms have several advantages that make them of interest for this dissertation. First of all, they require a minimum amount of problem information, as they do not rely on gradients for search and only need a single fitness value for a proposed solution [36]. This makes the algorithms able to handle non-differentiable search spaces that are challenging for standard, gradient-based NLPs solvers. Next, they have genetic mutation-emulating operators that, in theory, allow these algorithms to escape locally optimal solutions in search of globally optimal ones [33]. This is especially helpful for problems that are highly multimodal with multiple, locally optimal solutions. Another great advantage of evolutionary

algorithms is that they are not as sensitive to an initial guess as many standard gradient-based solvers are [33]. In fact, they typically begin with a randomly selected starting point (or set of starting points) [36]. Next, these evolutionary algorithms are able to solve black-box optimization problems where little information is known or needed regarding the problem itself. A performance metric, or fitness value, that is returned from a given input is all that evolutionary algorithms require. This can be very useful for problems that consist of complex models where gradient information is not readily available [41]. In addition, there are company proprietary and classified information issues that can be avoided given that these algorithms only require that a function can be evaluated for a given trial solution. Full disclosure of the model needed to determine associated gradient information is not required. Lastly, evolutionary algorithms are notoriously well-suited for parallel computation [42], [43]. As will be described later in Section 3.17, evolutionary algorithms require a minimal amount of information about the problem and operate on sets of candidate solutions making them perfect for parallel computing applications.

Unfortunately, these algorithms are not without their own unique set of shortfalls when compared to more traditional gradient-based NLP solvers. Evolutionary algorithms are not inherently setup to handle constraints, thus requiring special techniques such as penalty functions or repair methods [33]. This can cause difficulty, especially with problems that have large numbers of equality constraints, such as the ones resulting from collocation methods. In addition, evolutionary algorithms often take longer than gradient-based techniques to converge to a solution given that they are not leveraging gradient information for the given problem [33]. Next, these algorithms do not always converge to solutions as accurately as their gradient-based counterparts [33]. Furthermore, there is not a useful proof of convergence for these evolutionary techniques [44]. In addition, they typically do not utilize necessary conditions that can be helpful in determining convergence criteria, making it difficult to know exactly how many iterations these algorithms should perform to reach a solution. Next, these evolutionary algorithms require a number of tunable, user-defined parameters that can significantly affect how well the algorithm performs [33]. Unfortunately, the tuning of these parameters is a problem-dependent task without a well-defined method for optimally tuning them across all types of problems.

The drawbacks associated with evolutionary algorithms become the driving force behind the research conducted in this dissertation. All of the studies and contributions outlined in

this work have the underlying purpose of addressing three overarching goals associated with these algorithms. The first goal is to explore and implement effective constraint handling techniques that enable these algorithms to find feasible solutions to constrained optimization problems. The next goal is to improve the computational speed and efficiency associated with the various forms of evolutionary algorithms. Lastly, this dissertation develops techniques to address the goal of consistently obtaining sufficiently accurate solutions within a finite number of iterations. These three goals define the direction and overall purpose for the remainder of this dissertation.

1.4 No Free Lunch Theorem For Optimization Algorithms

Wolpert et al. develop several no free lunch theorems for optimization algorithms in general that are applicable to this dissertation and should be kept in mind when comparing various optimization algorithms [45]. Throughout several of the chapters in this dissertation, various optimization algorithms are compared on sets of benchmark functions. Wolpert et al. use Bayesian statistical approaches to prove that all algorithms perform equally when the entire set of all possible optimization problems are considered. While this is mathematically proven in their paper, Wolpert et al.'s words are sufficient for the purposes of this dissertation:

This theorem explicitly demonstrates that what an algorithm gains in performance on one class of problems is necessarily offset by its performance on the remaining problems; that is the only way that all algorithms can have the same f-averaged performance. [45]

With this in mind, it is important to realize that the comparisons between various optimization algorithms conducted in this dissertation can only illustrate how well the various algorithms perform on the specific class of optimization problems being considered. Claims can not be made about which algorithm is better for all types of problems. Based on the No Free Lunch theorems for optimization algorithms, all of these algorithms are equal when considering all possible functions. However, performance metrics over certain classes of problems that are of interest to this research are able to give insight into how the various algorithms perform relative to one another on these specific classes of functions. Throughout this dissertation, a number of continuous (real-valued), highly multimodal functions are

investigated and used to evaluate algorithm performance. While this class of functions outlined in this dissertation are not combinatorial and have real-valued variables, they are not all continuously differentiable. In addition, optimal control problems are also investigated that result in constrained, real-valued NLPs. These are problems that are often considered to be challenging to solve and are of great interest to the aerospace community. It must be noted that while many of the evolutionary algorithms developed in this dissertation perform well on these classes of problems, this is not necessarily the case for other problem types. Moreover, an algorithm tuned for a specific class of problem may not be tuned well for other classes of problems. Given this theorem, this dissertation is able to draw conclusions of algorithm performance and applicability that are specific to classes of problems that are of particular interest within the aerospace engineering community.

1.5 Dissertation Research Contributions

The overarching purpose of this research has been to develop modified evolutionary algorithms that are able to achieve, to the greatest extent possible, the following goals: effective constraint handling, improved computational speed, enhanced algorithm efficiency, and the ability to consistently find accurate solutions to problems with a finite number of iterations. The following list of contributions represent new developments from this dissertation that have been shown to directly (or indirectly) address these top-level goals surrounding evolutionary algorithms.

Parallel RLV Landing Footprint Determination

An application of parallel computation is used to rapidly generate RLV landing footprints. Given that parallel computation is directly applicable to evolutionary algorithms, the issues related to parallel computation are studied using a relevant example problem in aerospace . Useful insights of appropriately balancing computational workload coupled with parallel processing memory architectures are useful in the context of this real-world RLV optimal control problem. It must be noted that workload balancing and memory architecture studies have been done extensively, however, it is unique to study them through the lens of an RLV optimal control engineering application. Some of the results of this work have been published in the form of a conference paper [46].

Parallel GA Exact Penalty Function Analysis

A unique trade study is used to explore the performance of exact penalty functions within the context of parallel GAs. This directly applies to the challenge regarding the ability for evolutionary algorithms to efficiently handle constraints. It is discovered that the exact penalty functions, though not as frequently applied in GAs, seem to produce better results when compared to inexact forms of penalty functions. This is worthwhile to note, given that evolutionary algorithms do not require continuously differentiable functions and therefore are able to work with exact penalty functions. This study is unique in that it performs trades on various forms of exact and inexact penalty functions to an extent that has not been done with parallel GAs.

Parallel GA Type and Optimal Control Trade Study

Six different parallel GA types are studied as they apply to a lunar lander optimal control problem using exact penalty methods. Trades are conducted on parameters that adapt the parallel architecture of these various forms of GA. It is shown that parallel processing not only improves accuracy and convergence speed, but also significantly improves algorithm robustness, in regard to otherwise sensitive user-defined parameters. Much of this work has been published in the form of a conference paper [47].

Dynamic real-coded genetic algorithm (RCGA) Crossover Operators

In an attempt to further improve algorithm accuracy and efficiency, two new Gaussian-based dynamic covariance crossover operators are introduced and developed (dynamic truncated Gaussian crossover (DTGX) and dynamic Gaussian crossover (DGX)) while conducting a unique study of parallel RCGAs as they apply to optimal control [47]. The new operators are developed to introduce several characteristics that are not typically found in RCGA crossover operators. More specifically, they utilize Gaussian distributions in a way that favors solutions with better objective function values and have the ability to vary the search resolution by changing the distribution covariance over time. These new crossover operators achieve similar performance when compared to state-of-the-art RCGA crossover operators. This exercise proves to be valuable given that it provides useful insights and lessons that help lead to the introduction of an unscented RCGA.

Unscented Sampling In RCGAs

A new form of RCGA is introduced that utilizes deterministic sampling instead of the

standard random sampling techniques. This unscented RCGA proves to be a critical stepping stone for the development of the various ESs that utilize unscented sampling techniques. It is empirically demonstrated that this new form of unscented RCGA is able to achieve very accurate results (within $1.9 \times 10^{-3}\%$ of the known optimal solution) in solving a lunar lander optimal control problem when run on multiple processors in a parallel island architecture. This technique is shown to help address the challenges associated with computational speed and solution accuracy.

Unscented Sampling In ESs

ESs were found to be a more natural fit for the application of unscented sampling when compared to standard RCGAs. With this, a transition is made and the idea of unscented sampling is applied to the construct of an ES. Initially, a fairly simple ES is developed by building upon the idea of the DGX crossover operator. The unscented simulated annealing evolution strategy (USA-ES)/Gauss-Hermite simulated annealing evolution strategy (GHSA-ES) are created by applying a DGX-inspired mutation operator that uses unscented sampling within the general construct of an ES. Empirical analysis shows that this new algorithm obtains enhanced performance in terms of accuracy and computational time when compared with a similar ES that utilizes Monte Carlo sampling. These new unscented ESs prove to be very effective on many of the highly multimodal benchmark test problems as well as the lunar lander optimal control problem. Both 3rd order unscented sigma points and higher order sparse Gauss-Hermite points are used within this algorithm construct. These new ESs that use deterministically chosen points are additionally shown to be competitive with the state-of-the-art covariance matrix adaptation evolution strategy (CMA-ES). A parallel island implementation of this new USA-ES is developed to further enhance algorithm performance. It is then applied to the lunar lander problem as another unique contribution and proof of concept that is able to produce very accurate solutions (within $7 \times 10^{-5}\%$ of the known optimal value). All of these results indicate that this contribution may prove to be a very useful in addressing many of the challenges regarding evolutionary algorithms discussed above. A portion of this work has been published in the form of a conference paper [48].

Unscented Sampling In The natural evolution strategy (NES)

The sophistication and performance of utilizing unscented sampling within the context of an

ES is further increased by integrating this idea into a new state-of-the-art NES. This allows for the full utilization of the unscented abscissas where both the nodes and the associated weights are used to evolve both the distribution mean, and the distribution covariance. This is a major, unique contribution that has shown great promise in solving the challenging set of benchmark problems from literature. Empirical results indicate that this new unscented exponential natural evolution strategy (uxNES) is able to find globally optimal solutions for the many of the multimodal problems. Lastly, this new uxNES written in MATLAB code, produces near-optimal results for a lunar lander problem much faster than other evolutionary methods investigated without the use of parallel processing.

uxNES Global Optimization Algorithm

This new uxNES algorithm is then integrated into a global optimization scheme using the concepts of a reigning optimal and fitness scaling to further enhance the capabilities of the uxNES. The resulting algorithm addresses the goal of improving the algorithm's ability to consistently find accurate solutions within a finite number of iterations. This is a unique contribution in and of itself that produces promising empirical results warranting further research and development.

1.6 Dissertation Overview

The remainder of this dissertation begins with an overview of the computational hardware and programming techniques typically used for parallel computing. In conjunction with this, an illustrative example of applying parallel computation to quickly determine RLV landing footprints is conducted in Chapter 2. This example uses a commercially available optimization software package to illustrate how parallel computation can be used to enhance computational time associated with solving fairly complex astronautical engineering problems. The RLV application is used to identify trades associated with parallel programming that directly apply to the parallel evolutionary algorithms developed throughout this dissertation.

From this point on, the dissertation exclusively focuses on evolutionary algorithms. Chapter 3 provides the reader with an in-depth literature review on GAs where both the standard binary-encoded GAs along with RCGAs are developed and presented in great detail. A number of theoretical aspects to include various advantages and disadvantages associated

with the various types of GAs are discussed at length. Current state-of-the-art techniques associated with several different GA types are also covered. In addition, this literature review investigates various methods and program architectures used for running these types of algorithms in parallel. All of this information is utilized as a foundation that is then built upon to investigate new areas and develop new techniques associated with GAs.

Chapter 4 introduces and develops unique contributions specific to GAs and RCGAs as they apply to optimal control. First, a unique study is conducted to investigate various forms of exact penalty functions and their relative performance on parallel GAs within the context of optimal control (Section 4.4). This trade study is done to find a technique that can address the goal of achieving effective constraint handling within evolutionary algorithms. Next, two new and innovative RCGA crossover operators that utilize normal distributions with dynamic covariance values are developed and applied to a lunar lander optimal control problem (Section 4.5.2-4.6). Lastly, it is determined that the use of parallel processing on RCGAs can significantly alleviate the sensitivity of user-defined, tunable parameters and increase algorithm robustness, directly addressing several of the shortfalls associated with these types of algorithms.

Chapter 5 provides the reader with an in-depth overview and literature survey of the various ESs. This sets the foundation that is then built upon in the remainder of the dissertation. It is standard practice for ESs to use random sampling from Gaussian distributions to explore a search space [49]. After reading a number of papers and articles on ESs, it quickly becomes apparent that the new Gaussian-based crossover operators, developed earlier in this dissertation, are a perfect fit within the construct of an ES.

Chapter 6 introduces new forms of ESs that utilize unscented sampling techniques to more effectively explore search domains. This new form of ES is further developed and applied to a series of multimodal test problems throughout Chapter 6 where it is compared to identical ESs that use random sampling to isolate and determine the direct benefits of unscented sampling methods. This new ES is then shown to outperform state-of-the-art ESs on a number of benchmark test problems found throughout ES literature. Lastly, a parallel version of this new unscented ES is applied to lunar lander optimal control problem, producing very accurate results. These contributions are shown to directly address the afore-mentioned NLP challenges associated with multimodality and initial guess sensitivity.

From this point, the dissertation takes the application of unscented sampling techniques within the construct of ESs to an increased level of sophistication where the deterministically chosen sample points and associated weights are fully utilized in the search process. Chapter 7 outlines the method of employing unscented sampling techniques to more accurately calculate search gradients that are used to guide a new form of ES toward more promising regions of the search domain. This new form of unscented ES results from incorporating deterministic sampling and quadrature techniques into the construct of a NES. This NES algorithm is a new, state-of-the-art ES that was introduced by Wierstra et al. in 2008. [50] A unique and in-depth analysis of search gradient accuracy with regard to various sampling techniques is conducted, prior to implementing these ideas. The new unscented version of the NES is put to the test on a challenging suite of multimodal benchmark problems from literature where it is compared with other state-of-the-art ESs. This unscented NES proves to be very effective at solving these challenging test problems. It is also shown to be effective in finding near-optimal, feasible solutions to the lunar lander optimal control problem. Lastly, this research incorporates this newly developed ES into a global optimization algorithm to further improve its performance, resulting in another unique contribution from this work. The new techniques and algorithms developed in this chapter also address the challenges associated with multimodal problem optimization, along with the sensitivity of algorithm performance with regard to the initial guess that many NLP solvers face.

Lastly, this dissertation concludes with a discussion of several relevant astronautical engineering applications that the newly developed evolutionary algorithms developed in this work are particularly well-suited for. In addition, a number of future work ideas and proposed improvements to the unscented RCGA and the uxNES are introduced and discussed. It must be noted that a chapter-by-chapter definition of symbols is included in Appendix A. Next, Appendix B includes an extensive survey of various constraint handling techniques that have been developed and used in evolutionary algorithms. Finally, a new unscented RCGA is introduced and described in Appendix C. This new unscented RCGA is placed in the appendices to avoid confusion and allow for a more logical transition to unscented sampling techniques within the construct of evolutionary algorithms. A parallel version of this new RCGA is developed and applied to a lunar lander optimal control problem, producing very promising results. While this new sampling technique has never been used within the construct of a RCGA and is a unique contribution by itself, it plays an even

more important role as a critical stepping stone to other unique and innovative contributions associated with ESs.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 2: Parallel Computing Concepts

2.1 Introduction

Parallel computing concepts are used throughout this dissertation to enhance optimization algorithm performance in terms of both accuracy and convergence speed. Prior to developing and applying new optimization algorithms, a top-level introduction to the various forms of parallel computing is warranted. While the parallel implementations discussed in this dissertation only utilize a small subset of the parallel computing techniques and hardware, it is worthwhile to discuss these ideas in a broader sense to identify potential options for future work. Even though several of the techniques described in this chapter are not directly applied, they are included to help the reader understand where additional improvements in terms of computational speed might be achieved. While much of the research in this dissertation is focused on new techniques and modified optimization algorithms, it must be noted that the application of parallel processing to existing algorithms and engineering problems can be very rewarding. This chapter illustrates this through the use of a real-world engineering problem where commercially available optimization software is applied in parallel to significantly decrease computation time required to obtain a solution. Even within the context of this simple application, it will be discovered that there exist many trades and factors that need to be considered in order to realize good performance when using parallel computation.

This chapter begins by discussing the recent shift toward parallel computation on mainstream devices. After this, various hardware platforms and program architectures are discussed at a high level to shed light on the potential options that exist when parallel computation is desired. Next, parallel computing in an optimal control application is investigated where the time required to calculate an RLV landing footprint is significantly decreased through parallel processing. This problem, while somewhat complicated in terms of the dynamic models and physics involved, turns out to be very well-suited for parallel computation. This example offers a very relevant platform to highlight some of the hardware and software choices that need to be considered when solving problems with parallel computation meth-

ods, even when working with perfectly parallel problems. This concept can affect multicore processor performance, as many chip designs utilize various shared cache memory architectures as a way to maximize efficiency on multicore processors. However, this does require additional synchronization between cores to enforce memory coherency [51].

2.2 Shift Toward Mainstream Parallel Processing

It is important to understand that a revolution in computing hardware has taken place over the past 15 years. This rapid change in central processing unit (CPU) design is due to physical limitations that have been reached by chip manufactures as they have continued to pursue faster clock speeds and memory access times. Unfortunately, these physical limitations cannot be directly overcome by current processor technologies. This change from standard CPU designs is one that revolves around a shift toward parallel processing, concurrency, and multi-core CPUs on personal computing devices. This trend is a direct result of chip manufacturers reaching physical limitations in the early 2000's that quickly resulted in bounds on CPU clock speeds and memory bandwidths [51]. Chip designers had previously been following Moore's Law by increasing transistor densities within single core CPUs so that the number of transistors could be increased at exponential rates without significant growth in processor size. Up until the early 2000's this strategy produced great results in the realm of processor clock speed improvements which led to superlinear increases in CPU performance [51]. Unfortunately, as transistor thicknesses continued to decrease, the amount of current leakage and associated heating within the CPU increased to the point where elaborate cooling mechanisms quickly became necessary for chip makers to continue down the path of increasing transistor densities to achieve faster clock speeds [52]–[54]. In order to overcome the clock speed stagnation, CPU designers found a way to sidestep the issue by utilizing concurrency and multi-core processing in a way that effectively put the burden of improvement back on the shoulders of software developers [53]. This is why multi-core processors have become commonplace in personal computers, laptops, tablets, and even smart phones. Rather than increasing clock speeds, improvement in computational speed could continue through the use of parallel processing. In fact, on some modern computers, clock speeds have been slightly reduced with the use of multiple cores to preserve power and limit battery usage while still achieving high performance through the effective employment of parallel computation. Given this recent trend, it is not

beyond the realm of possibility that the next generation of space systems could leverage parallel computing devices on multiple levels. While this shift in CPU architectures has occurred fairly rapidly, the adjustments, redesign, and new way of thinking in the software and application development arena is an ongoing process that is still in its infancy [54], [55]. It is for these reasons that parallel processing is investigated and employed, when applicable, throughout this dissertation research.

Another physical limitation that can quickly become the computational bottleneck and limit CPU performance is the time required for processors to access the various levels of memory [56]. Memory access times have not been decreasing as fast as clock cycle times. As an example, access to main memory took an average of 6 to 8 clock cycles in 1990 and it took an average of 100 and 250 clock cycles in 2006, indicating that clock speeds have increased faster than memory access speeds [56]. This limitation basically comes down to the physical limitations of how fast electrons can travel over various lengths of conductive material. In order to address this issue, chip manufacturers have been working to maximize the amount of on-chip memory referred to as cache memory, so that shorter distances and faster read/write times between the processor cores and memory can be achieved [56].

2.3 Hardware

This section discusses the typical computational hardware configurations and components that are often used for parallel computation. This is not meant to be an in-depth analysis, but rather a top-level overview to familiarize the reader with the various options available for exploitation by parallel programs. In many cases, several of these methods can be used concurrently.

2.3.1 Parallel Processing Within A Single Processor

Prior to discussing multi-core CPUs and cluster computing, the parallel processing within a single processor is first introduced and discussed. The first method of increasing the amount of parallelism within a chip is often referred to as pipelining, or instruction-level parallelism [56]. This concept utilizes an assembly line type approach where different operations can occur on multiple instructions concurrently. Figure 2.1 provides a visualization of this idea. In this example there are four unique tasks and four different instructions being processed

at a time by the processor. Each of the four tasks can only operate on a single instruction during each clock cycle, however, the instructions have been staggered in a way that allows for maximum productivity, where multiple tasks can be conducted concurrently on different instructions. In Figure 2.1, each column represents a clock cycle and the center column represents a clock cycle where all four operations are able to work concurrently on all four of the instructions. The efficiency of this process is dependent upon the inter-dependencies that exist between instructions [56].

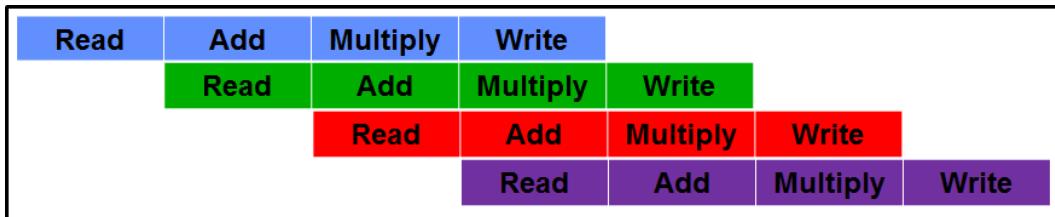


Figure 2.1. Visualization of pipelining

Another form of instruction-level parallelism can occur when chips have been designed to have redundant functional units [56]. These chips have the ability to perform the same task simultaneously on different instructions. As an example, a chip might have multiple Arithmetic Logic Units so that these functional units could be performing the same operations on different instructions within the same clock cycle [56]. Lastly, there is also the idea of data-parallelism or vectorization where special compilers can utilize functional units have that the ability to perform an operation on multiple bits during a single clock cycle [57]. This method can lead to significant speedup on certain applications that perform any number of operations that are able to be vectorized, such as the addition of two large arrays. While the previous two concepts often occur fairly autonomously, as far as the programmer is concerned, the vectorization of operations often requires specialized CPUs and compilers that often need to be told which portions of the code to perform vectorized operations on.

2.3.2 Multi-Core CPUs

As indicated earlier in this chapter, a fairly recent revolution in parallel computing has been occurring where multi-core CPUs have become mainstream and are now standard on many personal computing devices. A multi-core CPU consists of a single processor chip that

has multiple processor cores which are able to handle separate threads or control flows, sometimes referred to as thread-parallelism [58]. Whereas many of the instruction-level parallelism ideas are automatically handled at a lower levels, multi-core CPUs and the idea of multiple threads require special programs written in a way that specifically tasks different CPU cores with different jobs or control flows to execute. The multi-core processor is the primary hardware unit that is utilized throughout this dissertation for many of the parallel computation tasks. Given that multiple cores are sharing a single processor chip, there is often times the requirement for these cores to share on-chip cache memory. There are a number of different memory architectures that can be used with multi-core CPUs. One of the more common memory architectures is referred to as a hierarchical architecture. In this type of memory architecture, the fastest and smallest on-chip memory caches (referred to as L1) are shared by a fewer number of cores than the larger, slower L2 and L3 caches [56]. While there are different memory architectures, there is typically some degree of shared memory and required read/write synchronization that must occur between cores on many multi-core CPUs. This coordination between cores to enforce memory coherency can increase the communication costs as the number of cores sharing a memory cache grows [51]. This aspect will be illustrated in Section 2.8.

2.3.3 graphics processing units (GPUs)

GPUs are not applied in this dissertation, but are introduced and referenced as they offer the potential for additional computational speed and improvement for some of the optimization algorithms and architectures investigated in this research. These devices are often distinguished from CPUs in that they can have hundreds of cores and handle hundreds of threads at a time; however, GPUs can come in many different forms and architectures [58]. For the purposes of this brief introduction, it will be assumed that GPUs have a large number of simplified cores that can only perform a subset of operations as compared to cores that one might find on a multi-core CPU. This enables them to conduct very large numbers of fairly simple operations quickly. The GPU architecture lends itself very nicely to some applications and programs, while it may not be as practical for others. Typically programs that work with large data structures and have lower interdependencies between threads lend themselves well to GPU computing more readily than programs operating with smaller data structures and higher frequency communication steps.

2.3.4 Cluster Computing Devices and Hamming

Cluster computing refers to a number of computers, or compute nodes, that have their own unique processor components and local memories. Each node could contain a single-core processor, a multi-core processor, or other computing devices such as GPUs. These nodes are all interconnected through any number of various network topographies. These individual nodes typically have private memory and data can be passed back and forth using the network connections and a message passing interface (MPI) [56]. Cluster computing devices often times share a common operating system and require a scheduler to assign processes to computational resources. Many of the parallel algorithms used throughout this dissertation were run on the Naval Postgraduate School (NPS) Hamming supercomputer.

The Hamming hybrid-cluster supercomputer was named after the famous mathematician and former NPS instructor Richard Hamming [59]. The Hamming computer has a total of 3,178 Intel and AMD processor cores with an additional 6,304 GPU cores located on 49 different compute nodes [59]. Hamming is constantly being upgraded, and these numbers are subject to change. The interconnections between the various nodes utilize Infiniband high speed connections capable of handling up to 32 gigabytes of data per second [59]. The current theoretical peak performance of Hamming including GPUs and hyperthreading on special, multi-integrated cores (capable of executing two threads at a time) is 36.0753 trillion floating point operations per second. [59] Lastly, Hamming utilizes a CentOS 6.4 operating system with a 64-bit Linux kernel. While there are GPUs on Hamming, all of the parallel computing work done on Hamming in this dissertation is conducted on standard AMD or Intel multi-core CPUs utilizing a combination of the C programming language and MPI or MATLAB's built in parallel toolbox.

2.4 Program Architectures

Flynn's taxonomy of programming architectures is often used to describe the top-level classifications of hardware and program control flow [56]. The most basic programming architecture identified in this taxonomy is the Single Instruction Single Data concept. This is where a single program sends instructions to a single processor that reads data from memory, executes the instructions in serial, and then writes the result back to memory. This is the standard architecture for serial computing. The single instruction multiple data (SIMD) program architecture consists of multiple processing elements reading a single instruction

that is then executed in parallel. An example of this could be the previously mentioned vectorized operations where multiple data elements of a large array can be divided among processing elements and operated on in parallel. GPUs were originally used for these types of applications relative to pixel vectors used in graphics processing [58]. A third class of multiple instruction multiple data (MIMD) program architectures are the most commonly used type of parallel program architectures and are often applied to multi-core and cluster computing devices [56]. There are many various forms of this architecture, however, single program multiple data (SPMD) is the one used in the parallel algorithms explained in this dissertation. SPMD uses a single program that divides up computational work by sending out different instructions to the different processing elements. This is often done by utilizing unique identification numbers that are assigned to each processor so that the program can identify and send specific instructions to certain processors that can then work in parallel, asynchronously with respect to one another. This also allows the processors to communicate back and forth with specific processors, which makes it possible to develop any number of communication networks used to transfer data between processors.

2.5 Parallel Footprint Generation For RLVs

At this point, a relevant example is used to illustrate the potential benefits and sensitivities associated with an application of multi-core parallel computation in a realistic astrodynamical engineering problem. There are many lessons and important trades that will be taken from this example problem that are applied later in the parallel evolutionary algorithms explored throughout this dissertation. While this application problem utilizes a commercially available optimal control software package, it will be seen that many of the parallel computing considerations are applicable to evolutionary algorithms as well. This study on parallel RLV footprint generation was originally presented and published in the 2015 AAS/AIAA Astrodynamics Specialist Conference in Vail, CO [46].

There are a number of engineering processes, models, and calculations that require a series of independent tasks to be solved and pieced together to create a final solution. In the light of applying parallel computing to computationally expensive engineering problems, these types of processes are the low-hanging fruit just waiting to be harvested. This class of problems that are composed of a series of independent tasks are often referred to as “perfectly parallel” problems because little to no communication is required between

processors during the execution of the program. Admittedly, RLV footprint generation directly falls into this category and is a perfect real-world engineering problem to illustrate useful applications of multi-core computing in optimization and optimal control problems.

RLVs have been increasingly considered for launch applications due to their potential to significantly reduce launch costs. Before these cost savings can be realized, future RLV systems will need to attain sufficient launch frequency while simultaneously achieving increased levels of automation, safety, and reliability [60], [61]. The ability to determine a landing footprint, or maximum area on the Earth’s surface that a RLV could potentially reach as a function of its initial state, in near real-time, can help to increase flexibility and decrease decision times in determining the best landing zone to approach [62]. On heritage systems such as the Space Shuttle, a single optimal reference trajectory to a pre-selected landing site is typically calculated on the ground and then uploaded to the vehicle [5], [63]. While this method has proven to work for most scenarios, one can see how the capability to quickly calculate an entire series of trajectories in near real-time will open up a larger decision space and lead to increased safety, performance, and the ability to further reduce costs through automation. Contingency landing scenarios for manned RLVs are good examples of where near real-time footprint determination becomes extremely valuable for crew safety [62]. In these time-sensitive landing scenarios it becomes a necessity to have an accurate determination of reachable landing sites. It is understood that there are a myriad of potential anomalies and events that can lead to devastating results on all space systems. From an engineering standpoint, the best way to address these risks is to increase the redundancy, robustness, and recovery options for the given system. Advancements in relevant technologies often prove to be enablers for achieving these types of improvements on existing systems and concepts that were not originally envisioned by their designers and engineers. Given this, there have recently been several key shifts in computational technology that could prove be the enablers for improved performance and reliability on a number of systems to include RLVs.

This section investigates two different parallel schemes that utilize commercially available optimization software and a three degrees of freedom (3DOF) RLV dynamics model to illustrate how parallel computing can significantly reduce the computation time for calculations that have historically been time consuming. RLV footprint generation is a prime example of a task that is computationally expensive, when the footprint points are computed

in serial. This large time expense is the result having to solve a large number of optimal control problems at high temporal resolutions in order to generate accurate footprints. For serial computing, there is also a direct correlation between the number of points making up the footprint boundary and the amount of time it will take to complete the calculation. The results of this chapter will show that this is not necessarily the case with parallel computation.

2.5.1 RLV Model

This subsection describes the system dynamics model of the RLV used in this chapter. The RLV is treated as a point mass and a 3DOF dynamics model is used to describe its behavior as a function of control inputs during reentry into the Earth's atmosphere [62]–[65]. This model, described in Equations (2.1)–(2.6) is a simplification in that it assumes a spherical earth with a constant angular velocity and an inverse squared gravitational model $g(r) = \frac{GM}{r^2}$. These assumptions are reasonable given the purpose of this chapter which is to explain applications of parallel computing to the RLV footprint problem and the associated speedup. The state variables, $\bar{x} = [r, \mu, \lambda, V, \gamma, \xi]^T \in \mathbb{R}^6$, and control variables, $\bar{u} = [\alpha, \sigma]^T \in \mathbb{R}^2$, used in this model are defined in Table 2.2. In addition, Table 2.1 summarizes all of the constant values utilized in the RLV dynamics model, path constraints, and aerodynamic force equations.

$$\dot{r} = V \sin \gamma \quad (2.1)$$

$$\dot{\mu} = \frac{V \cos \gamma \cos \xi}{r \cos \lambda} \quad (2.2)$$

$$\dot{\lambda} = \frac{V \cos \gamma \sin \xi}{r} \quad (2.3)$$

$$\dot{V} = -\frac{D(\alpha, V, r)}{m} - g(r) \sin \gamma + \Omega^2 r \cos \lambda (\sin \gamma \cos \lambda - \cos \gamma \sin \lambda \sin \xi) \quad (2.4)$$

$$\begin{aligned}\dot{\gamma} = & \frac{L(\alpha, V, r) \cos \sigma}{mV} + \left(\frac{V}{r} - \frac{g(r)}{V} \right) \cos \gamma + 2\Omega \cos \lambda \cos \xi \\ & + \frac{\Omega^2 r}{V} \cos \lambda (\cos \gamma \cos \lambda + \sin \gamma \sin \lambda \sin \xi)\end{aligned}\quad (2.5)$$

$$\begin{aligned}\dot{\xi} = & \frac{L(\alpha, V, r) \sin \sigma}{mV \cos \gamma} - \frac{V}{r} \cos \gamma \cos \xi \tan \lambda \\ & + 2\Omega (\tan \gamma \cos \lambda \sin \xi - \sin \lambda) - \frac{\Omega^2 r}{V \cos \gamma} \sin \lambda \cos \lambda \cos \xi\end{aligned}\quad (2.6)$$

Symbol	Parameter	Value
m	RLV empty mass	2,455 slugs (35,828 kg)
S_{ref}	RLV reference area	$1,608 \text{ ft}^2$ (149.4 m^2)
σ_0	standard density	$0.002378 \text{ slugs/ft}^3$ (1.226 kg/m^3)
r_{ref}	reference altitude	20,902,900 ft (6,371.2 km)
β	inverse scale height	$4.20168 \times 10^{-5} \text{ ft}^{-1}$ ($1.38 \times 10^{-4} \text{ m}^{-1}$)
GM	Earth's gravitational constant	$1.4076539 \times 10^{16} \text{ ft}^3/\text{s}^2$ ($3.99 \times 10^{14} \text{ m}^3/\text{s}^2$)
Ω	Earth's angular velocity	$7.2722 \times 10^{-5} \text{ rad/s}$
R_e	Earth's radius	20,925,646.32 ft (6,378.14 km)
h_0	Initial RLV altitude	125,000 ft (38.1 km)

Table 2.1. RLV problem constants

2.5.2 Aerodynamic Forces

The aerodynamic forces are based on simple parametric functions that suffice for the purposes of this chapter. Wind currents are neglected and the simplified exponential atmospheric density model $\rho(r) = \rho_0 e^{-\beta(r-r_0)}$ is used. [62] In addition, Equations (2.11) and (2.12) provide representative 2nd order parametric fit curves that are based on X-33 data [66] and are used to calculate the drag coefficient, C_D , and lift coefficient, C_L . A 5th order curve fit is applied to standard atmospheric data and used to estimate the speed of sound as a function of altitude.

$$D(\alpha, V, r) = \frac{1}{2} \rho V^2 C_D(\alpha, M(V, r)) S_{ref} \quad (2.7)$$

$$L(\alpha, V, r) = \frac{1}{2} \rho V^2 C_L(\alpha, M(V, r)) S_{ref} \quad (2.8)$$

$$M(V, r) = \frac{V}{a(r)} \quad (2.9)$$

$$a(r) = -2 \times 10^{-24} h^5 + 2 \times 10^{-18} h^4 - 10^{-12} h^3 + 2 \times 10^{-7} h^2 - 0.0103 h + 1173.8 \quad (2.10)$$

where $h = r - R_e$

$$C_D(\alpha, M) = 0.0001432\alpha^2 + 0.00558\alpha - 0.01048M + 0.2204 \quad (2.11)$$

$$C_L(\alpha, M) = -0.0005225\alpha^2 + 0.03506\alpha - 0.04857M + 0.1577 \quad (2.12)$$

2.6 Generating RLV Footprints

Generating a RLV landing footprint requires solving a number of trajectories that begin from some given initial state and end at the furthest point (on Earth) that can be reached in a given direction. These trajectories can be solved using optimal control. The standard form of an optimal control problem is illustrated in Equation (2.13) [31]. In this formulation, \bar{x} is a state vector, \bar{u} is a control vector, $e(\bar{x}_f, t_f)$ is an endpoint manifold, and $h(\bar{x}, \bar{u})$ are path constraints. In addition to this, there are often upper and lower bounds on \bar{x} and \bar{u} . Relevant RLV footprint state and control variable definitions are summarized in Table 2.2 and their dimensions are defined in Equation (2.13).

$$\begin{aligned}
\bar{x} &= [r, \mu, \lambda, V, \gamma, \xi]^T \in \mathbb{R}^6 \quad \bar{u} = [\alpha, \sigma]^T \in \mathbb{R}^2 \\
&\left\{ \begin{array}{l} \text{Minimize: } J[\bar{x}(.), \bar{u}(.), t_f] = E(\bar{x}_f, t_f) + \int_{t_0}^{t_f} F(\bar{x}(t), \bar{u}(t), t) dt \\ \text{Subject To: } \dot{\bar{x}} = f(\bar{x}, \bar{u}, t) \\ \bar{x}(t_0) = \bar{x}_0 \\ e(\bar{x}_f, t_f) = 0 \\ h(\bar{x}, \bar{u}) \leq 0 \end{array} \right. \tag{2.13}
\end{aligned}$$

There are several ways that the cost function can be implemented to determine the RLV landing footprint. As an example, a simple weighted cost function $J(\bar{x}, \bar{u}) = \pm w \mu_f \pm (1 - w) \lambda_f$ where $w \in [0, 1]$ can be used to determine the perimeter of the RLV footprint. If $w = 0$ then the resulting trajectories will be the ones that have the maximum/minimum latitude at their respective endpoints. If $w = 1$, then the optimal trajectories will be the ones that have endpoints at the maximum/minimum longitude. As the weight value is adjusted from 0 to 1, the importance of maximizing/minimizing latitude vs. longitude will vary and the perimeter of the footprint can be traced out. One of the drawbacks to this method is that the resolution or spacing of the trajectories and final endpoints that define the perimeter of the footprint may not be uniform. Terminal points may group in certain areas while leaving other areas sparse. Figure 2.2 is an example of a footprint that has been generated using the aforementioned technique.

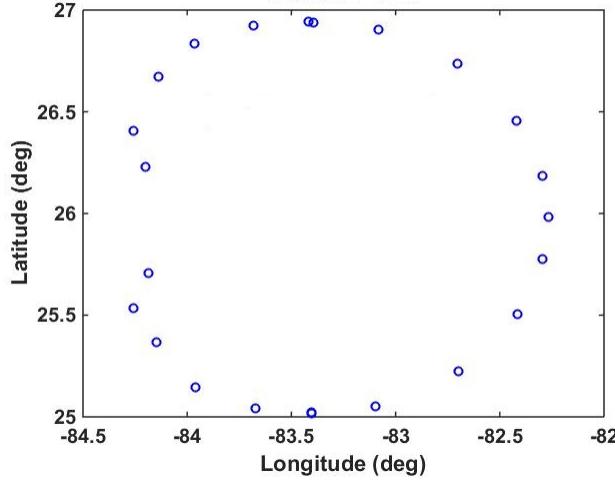


Figure 2.2. Example footprint points generated using weighted cost function method

Another approach is to use the cost function outlined in Equations (2.14) and (2.15). This objective function simply requests that latitude or longitude be minimized (or maximized). This simple cost function can be utilized in combination with endpoint constraints that specify feasible values of either μ_f or λ_f . In other words, the user is able find the maximum/minimum longitude for a set value of latitude, or vice versa. Feasible ranges of μ_f or λ_f are found in advance by running the four cases outlined in Equations (2.14) and (2.15) without enforcing any endpoint constraints on μ_f and λ_f .

$$\min J(\bar{x}, \bar{u}) = \pm \mu_f \text{ st: } \lambda_f = \lambda_{f \text{ desired}} \text{ where } \lambda_{f \text{ desired}} \in [\lambda_{f \text{ max}}, \lambda_{f \text{ min}}] \quad (2.14)$$

$$\min J(\bar{x}, \bar{u}) = \pm \lambda_f \text{ st: } \mu_f = \mu_{f \text{ desired}} \text{ where } \mu_{f \text{ desired}} \in [\mu_{f \text{ max}}, \mu_{f \text{ min}}] \quad (2.15)$$

These four cases of Equations (2.14) and (2.15) essentially find the maximum and minimum downrange and crossrange trajectories thus defining $[\lambda_{f \text{ min}}, \lambda_{f \text{ max}}, \mu_{f \text{ min}}, \mu_{f \text{ max}}]$. After these 4 values have been determined, all possible values of μ_f can be swept through while allowing λ_f to be free or vice versa. It must be noted that, due to the nonconvexity of the typical RLV footprint, the algorithm should sweep through both μ_f and λ_f to ensure that

Symbol	Description	Lower Bound	Upper Bound
r	radial position from Earth's center	R_e ft	$R_e + h_0$ ft
μ	geocentric longitude	-90 deg	90 deg
λ	geocentric latitude	-89 deg	89 deg
V	velocity magnitude	1.0 ft/s	17,060 ft/s
γ	flight path angle	-89 deg	89 deg
ξ	heading (clockwise from east)	-180 deg	180 deg
α	vehicle angle of attack	-10 deg	50 deg
σ	bank angle	-80 deg	80 deg

Table 2.2. Listing of RLV state and control variables and ranges

no portion of the footprint is missed, regardless of the RLV heading angle. This the results of this approach are further illustrated in Figures 2.3 and 2.4, where the triangle symbol indicated the direction that the RLV is heading.

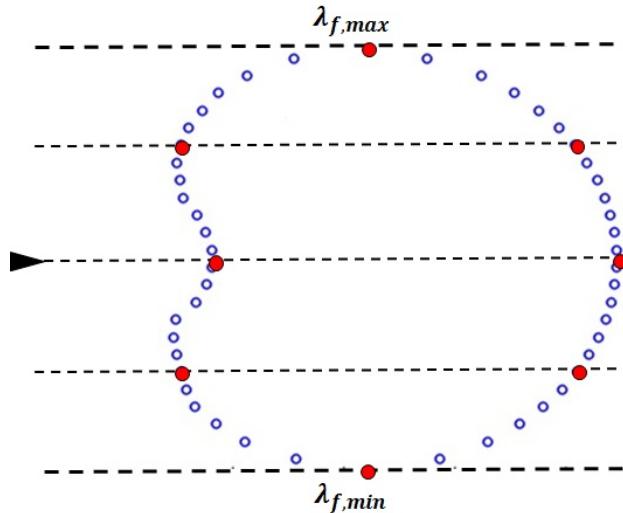


Figure 2.3. Example footprint points using latitude sweep method

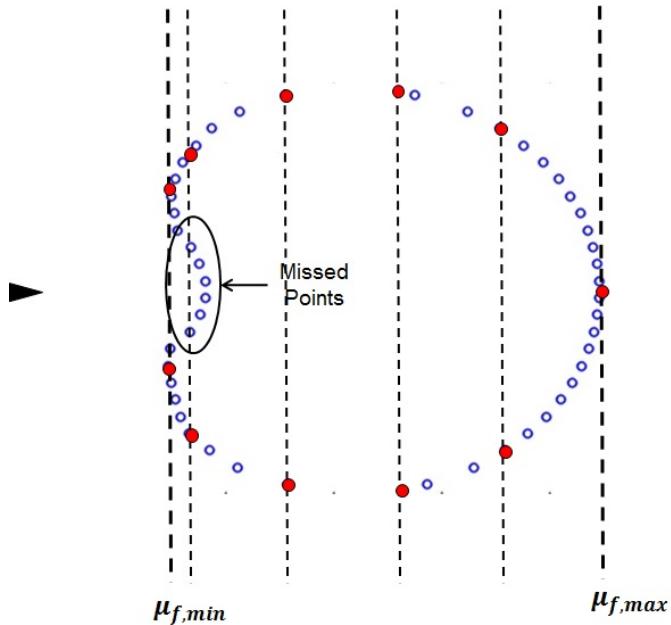


Figure 2.4. Example footprint points using longitude sweep method

Another approach would be to sweep through angles (or ratios of λ_f and μ_f) that define the bounds of the footprint to produce similar results. While this angle sweep approach only requires one sweep to reveal the entire footprint, it can be difficult to get good resolution on various edges of the footprint where these angular rays essentially become tangent to the footprint perimeter. In addition, the resolution is higher at the portion of the footprint that is closest to the RLV and the points become further apart as the distances increase. This aspect is further illustrated in Figure 2.5.

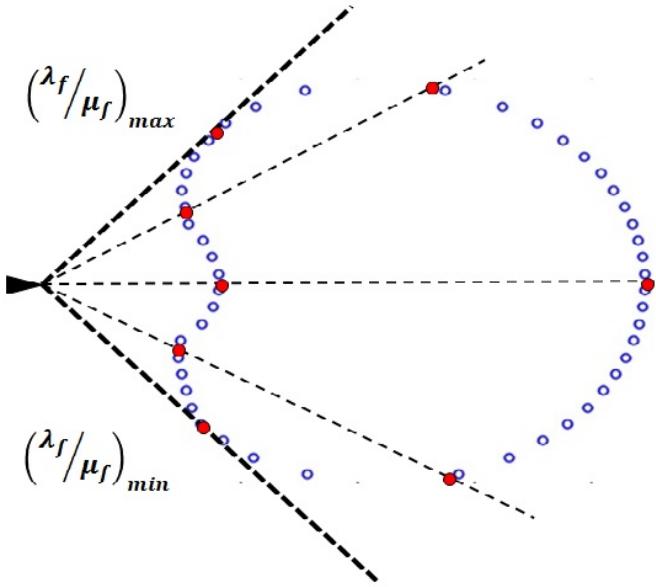


Figure 2.5. Example footprint points using angle sweep method

While any of the above approaches can work for RLV footprint determination, the latitude/longitude sweep method is used to produce the results for this chapter. All of the initial and final conditions that are used are summarized in Table 2.3. It must be noted that μ_f and λ_f can be free variables or fixed variables (as depicted in Table 2.3). This is dependent on whether fixed endpoint values on μ_f are enforced and λ_f is free or if end conditions on λ_f are enforced while μ_f is free. Given that the RLV has an initial heading that is due east, a simple λ sweep method can be used for the purposes of demonstrating the advantages of parallel computing techniques for the footprint generation problem.

The last part of the RLV reentry optimal control problem to discuss are the path constraints. These constraints are used to help engineers and system operators enforce a number of bounds on their system. These bounds can prevent flight into restricted air space or prohibit flight paths that could lead to members of the crew getting motion sickness. The path constraints outlined in Equations (2.16)-(2.18) are common path constraints that are often associated with these types of problems. The constraints enforce bounds on lateral g forces (η_Z), dynamic pressure (\bar{q}), and heat flux (Q). These path constraints are similar to ones

Symbol	Value
r_0	$R_e + h_0$ ft
r_f	$R_e + 500$ ft
μ_0	-85 deg
μ_f	free/fixed
λ_0	26 deg
λ_f	free/fixed
V_0	5,413 ft/s
V_f	335 ft/s
ξ_0	0 deg
ξ_f	free
γ_0	-1.3 deg
γ_f	-3 deg

Table 2.3. Initial/final conditions for RLV footprint problem

that would be encountered on vehicles such as the X-33, X-37, and X-40 [66].

$$\begin{aligned}\eta_Z &= L \cos \alpha + D \sin \alpha \\ -2.5g's &\leq \eta_Z \leq 2.5g's\end{aligned}\tag{2.16}$$

$$\begin{aligned}\bar{q} &= \frac{1}{2} \rho V^2 \\ 0 \leq \bar{q} &\leq 300 \frac{lb}{ft^2}\end{aligned}\tag{2.17}$$

$$\begin{aligned}Q &= k \sqrt{\rho} V^{3.15}; \quad k = 4.47228 \times 10^{-9} \\ 0 \leq Q &\leq 70 \frac{BTU}{ft-s}\end{aligned}\tag{2.18}$$

2.7 Parallel RLV Footprint Program Architecture

This chapter investigates two slightly different approaches for distributing the computational work required in footprint generation among multiple processors. While MATLAB is utilized for demonstration purposes, it must be noted that higher performance languages such as C, C++, or Fortran could be used for additional efficiency. Each of the proposed architectures use a MIMD programming model to achieve thread parallelism and run multiple instances of DIDO[©] on separate processors. DIDO[©] is a commercially avail-

able optimal control software package that is used throughout the aerospace industry and has been flight-proven by NASA on both the International Space Station [15] and on the TRACE Telescope [16]. The first approach, referred to hereafter as method 1, makes use of MATLAB’s parallel toolbox to parse out separate trajectory optimal control problems to each available processor through the use of parallel loops in accordance with Figure 2.6. The program initially determines the range that the footprint will span both longitudinally $[\mu_f \min, \mu_f \max]$ and latitudinally $[\lambda_f \min, \lambda_f \max]$ so that it can determine what conditions need to be enforced in order to generate the entire RLV footprint. From this point, the algorithm creates a user-defined number of equally spaced values between λ_{\min} and λ_{\max} (if μ_f is allowed to be free) or equally spaced values between $\mu_{f \min}$ and $\mu_{f \max}$ (if λ_f is allowed to be free). The algorithm used in method 1 then essentially tries to evenly distribute the work among processors in a way that allows each processor to compute the same (or close to the same) number of footprint points. It will be seen later on that this may not result in an even distribution of work given that each point can require a different amount of computation time. While MATLAB’s parallel toolbox makes it very easy to parallelize code, it can sometimes create difficulties associated with tuning and understanding the algorithm by masking the communication steps that are occurring under the hood. In method 1, MATLAB’s “parfor” or parallel for loop is used to automatically distribute iterations of the loop among the processors or “workers” as they are referred to in Figure 2.6. In addition to the parallelization occurring inside the parallel toolbox commands, newer versions of MATLAB automatically run certain commands with built-in multithreading to speedup execution times on multi-core CPUs. While this can increase the efficiency for many programs and is generally useful, it can also create difficulties in terms of understanding speedup curves and knowing exactly how many cores are being utilized for each case in the numerical experiments. It is for this reason that a second implementation is developed to remove some of these uncertainties.

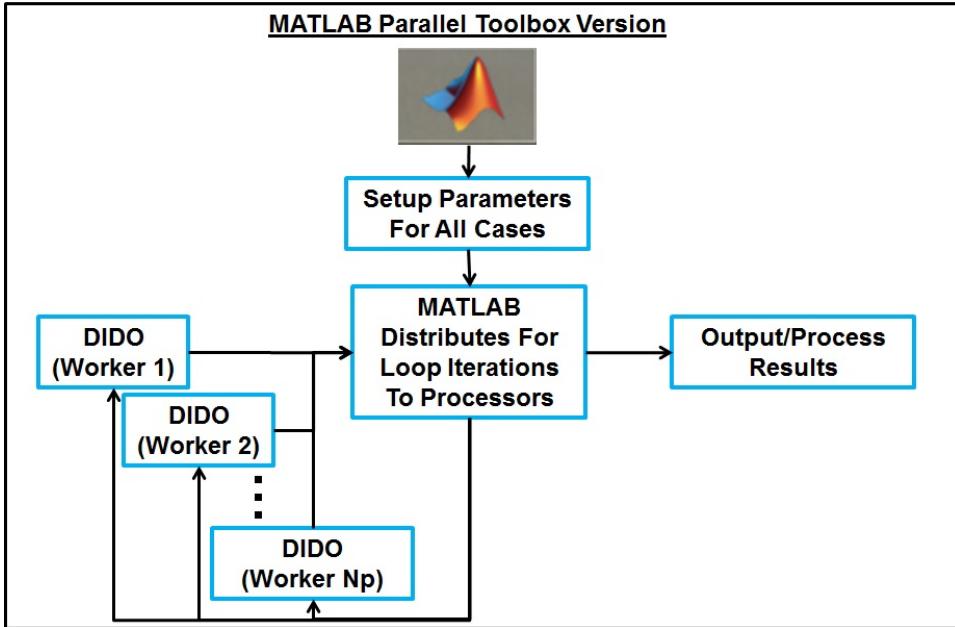


Figure 2.6. Method 1 RLV parallel program model (MATLAB Parallel Toolbox)

The second implementation depicted in Figure 2.7, referred to hereafter as method 2, is an explicit parallelization and begins with a single instance of MATLAB. This instance sets up an input file containing the various parameters and endpoint conditions for each trajectory that is to be optimized. Next, MPI is used to launch the requested number of processors, each of which startup a local instance of MATLAB. As part of the MATLAB startup command, each core passes MATLAB a flag (-singleCompThread) in order to avoid any unanticipated parallelization. This creates a more telling experiment and ensures that each instance of MATLAB only runs operations on a single thread. When using MPI, each processor is assigned a unique identifier ranging from 0 to $np - 1$ where np is the total number of processors. Using these unique processor identifiers along with np , each processor executes a simple algorithm to quickly determine which portion of the trajectories, or piece of the footprint that it needs to work on. This work sharing methodology splits up the RLV footprint in a way that ensures an even workload distribution in regard to the number of trajectories that each processor has to optimize. After the processor-specific work has been identified, each processor executes a serial “for” loop to run DIDO and obtain the trajectories in their respective portions of the footprint. Once each processor has solved all

of their tasked trajectories, they then output their results to an output file that is specific to that processor. The final step is for a single processor to compile all of the output files from each core into a single output file that is then plotted, analyzed, etc. This entire process is run from a single bash script file and all instances of MATLAB are launched in command line mode to minimize the amount of memory used. One major benefit with method 2 is the ability to run it on computing devices that have distributed memory architectures. While this is not demonstrated in this chapter, it will be seen later, that this would eliminate any computational expense that is caused by shared memory CPUs.

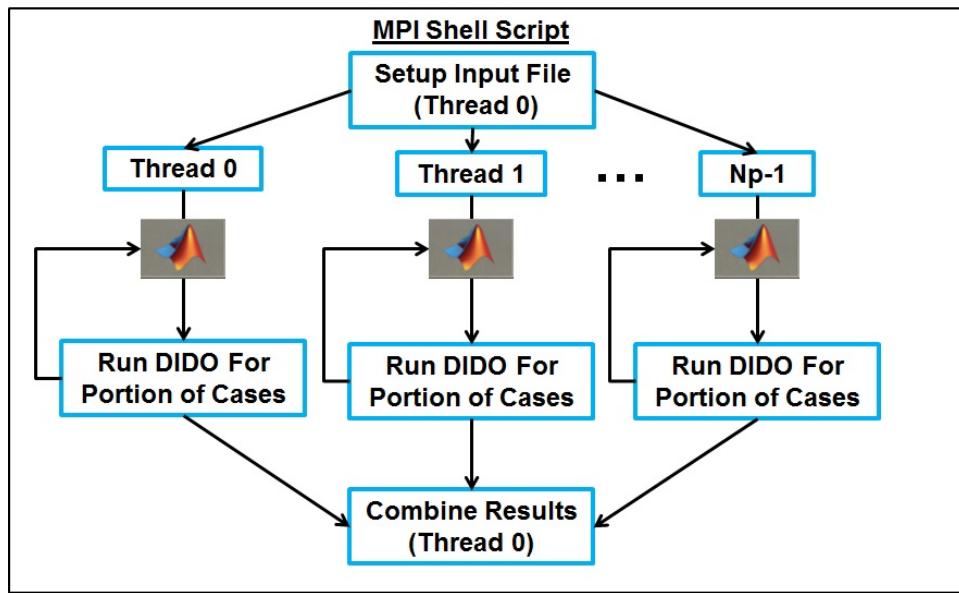


Figure 2.7. Method 2 RLV parallel program model (MPI shell script)

2.8 Parallel RLV Footprint Determination Results

All of the results in this chapter are obtained using a desktop computer with a 2.7 GHz, 12-core Intel Xeon E5 processor. Initially the RLV footprint is found with a standard, serial computation method. Figure 2.8 illustrates the resultant RLV footprint that is obtained. The same landing footprint is calculated in serial 10 times and the average time required to compute a single landing footprint is used as a benchmark for performance metrics.

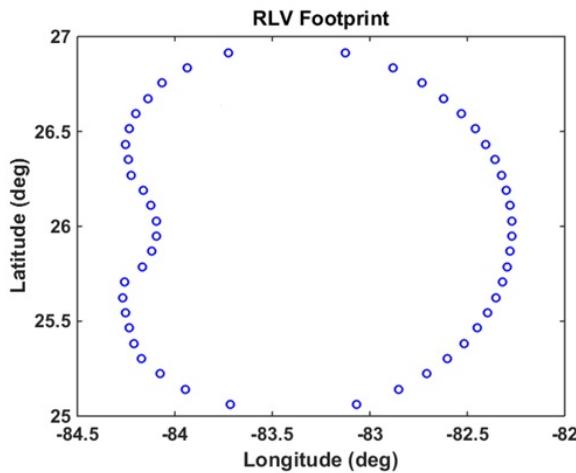


Figure 2.8. 48 point benchmark RLV footprint

Each trajectory or point on the footprint is also timed to see how uniform, or non-uniform the average required calculation time is for each point in the problem. Figure 2.9 illustrates that not all points around the perimeter of the footprint take the same amount of time to compute. This is important to keep in mind when determining how the computational workload is distributed among processor cores.

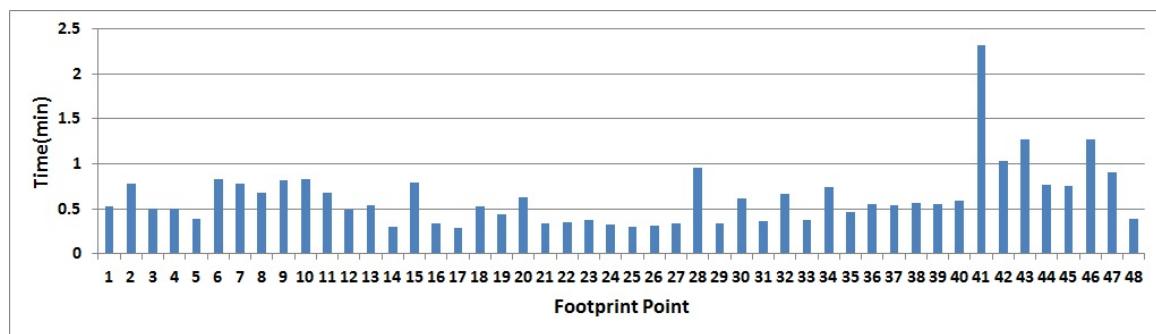


Figure 2.9. Solution times for each point on the benchmark RLV footprint

There are a couple factors that can cause the variations in trajectory computation time. First of all, these cases are initially computed using the guess-free mode of DIDO[©]. The software is then instructed to solve the problem using a fairly coarse grid with only 20 nodes. This coarse solution is then used as the guess for solving the same problem with 60 nodes for

higher resolution and accuracy. This method is sometimes referred to as bootstrapping [31]. The guess-free algorithm may be more accurate for some trajectories than others. Given this, some trajectories may be solved more quickly than others.

The second factor that contributes to the variance in computation time between points is problem scaling. For the RLV problem, different scaling factors are applied to partitioned regions of the footprint as illustrated in Figure 2.10. Scaling units are used in numerical methods to keep all values and parameters of the problem relatively close in magnitude. Significant differences in magnitude between variables can lead to slow convergence times, and even prevent convergence in some cases [62]. Canonical scaling units are commonly used for these types of problems due to their simplicity. Canonical scales are related to problem values and are often times functions of one another. As an example, for this type of RLV problem the canonical scheme outlined in Equation (2.19) could be used. In Equation (2.19) DU represents a distance scaling unit, MU is a mass scaling unit, TU is a time scaling unit, and VU is a velocity scaling unit. After the appropriate scaling units have been applied to the problem, all of the variables should be unit-less and of similar magnitudes. Unfortunately, for the problem outlined above, simple canonical units do not sufficiently scale the optimization problems for the conditions being used and designer scaling units are required. Designer units are simply customized units that are not necessarily related to each other, but can be independently tuned in a way that can effectively scale the problem.

$$\begin{aligned} DU &= r_0 \\ MU &= m \\ TU &= \sqrt{\frac{DU}{g}} \\ VU &= \frac{DU}{TU} \end{aligned} \tag{2.19}$$

In addition to designer units, partitioning is used to account for the fact that each trajectory, or group of trajectories requires a slightly different set of scaling units. Given the partitioned scaling method where each partition of the footprint is assigned different scaling units, it can be inferred that certain trajectories are better scaled than others. Scaling is one of the

more challenging aspects of solving these footprints in parallel, as it is difficult to find the appropriate designer scaling units a priori to attempting to generate the footprint. This is an area where additional research and potential automatic scaling techniques could be applied.

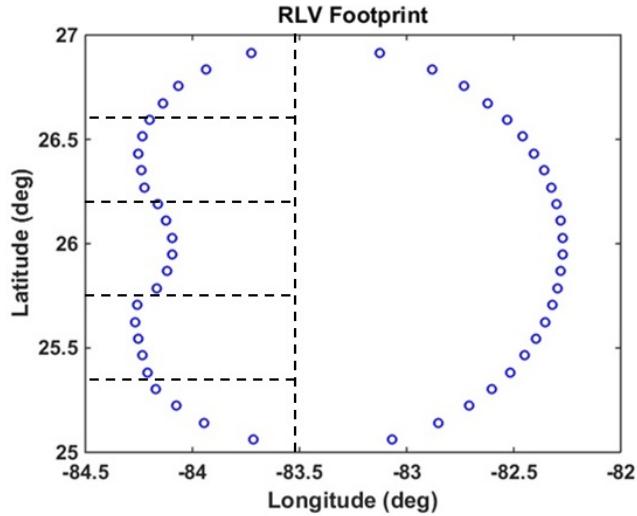


Figure 2.10. Example partitioned RLV footprint for scaling

Speedup is a common metric that is used when comparing various parallel computing methods to serial computation methods. Equation (2.20) relates the speedup observed for a problem of size n using p number of processors with $T^*(n)$ being the best known execution of a serial implementation of the code and T_p is the execution time in parallel [56]. Given the results obtained from the program architecture outlined in method 1, Figure 2.11 illustrates the speedup achieved as more cores are utilized. It can be seen that a significant level of speedup (about 4) is achieved through the use of this parallel processing program model. These curves are obtained by solving a 48 point footprint 10 times for each number of processors to get average computation times and speedup values for each case.

Theoretical speedup, S_p , can be a useful benchmark to help with the determination of parallel code efficiency. Amdahl's Law, illustrated in Equation (2.21), is one way to find this theoretical best speedup, given a constant fraction of the code that must be executed in serial $0 \leq f \leq 1$ [56]. This law essentially states that the code cannot run any faster than the time it takes to execute the longest serial portion of the code, regardless of how many processors are used. Amdahl's Law assumes that the code is perfectly parallel and

that there is no communication cost incurred due to the introduction of additional cores. Given the longest point in the above time distribution plot (f is approximately $1/13$ th of the total time), the best speedup that can be achieved with any number of processors is $S_{max} \leq \frac{1}{f} \rightarrow S_{max} \leq 13$. However, Amdahl's Law assumes that there is a series portion of the code followed by a parallel portion of the code that can essentially be parallelized down to zero. This means that, with enough processors, only the serial part of the code would require computation time. Due to this, Amdahl's Law does not exactly fit the algorithms in this chapter due to the fact that the RLV problem essentially has discrete blocks of time that cannot be broken down any further, but are still considered to be in the parallel portion of the code. For this chapter, linear speedup, will be treated as an upper limit or optimal speedup as the number of processors is increased until S_{max} is reached. This of course would only be achieved if the problem is perfectly parallel in that the work is evenly distributed among processors and there is no communication cost associated with multiple processors running in parallel.

$$S_p(n) = \frac{T^*(n)}{T_p(n)} \quad (2.20)$$

$$S_p(n) = \frac{T^*(n)}{fT^*(n) + \frac{1-f}{p}T^*(n)} = \frac{1}{f + \frac{1-f}{p}} \leq \frac{1}{f} \quad (2.21)$$

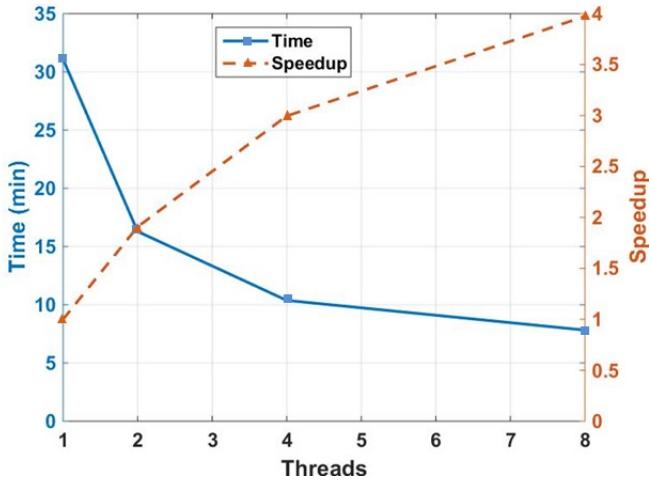


Figure 2.11. Speedup results for method 1 - shared memory (48 point footprint)

Given that footprint generation is pretty close to “perfectly parallel,” there are several potential causes that may be contributing to the discrepancy between actual and theoretical speedup values. It is known that the speedup is not exactly linear on multi-core CPUs that are cast on the same die with shared memory caches [51]. This is due to required coordination and handshaking that takes place between the cores on shared memory architectures to ensure cache memory coherency. The next potential cause could be due to the unequal times required to solve each trajectory. Recall that the optimal speedup values require that the work is evenly distributed among processors. This of course is not the case, given that both methods initially distribute the work by number of trajectories and not by execution time. As seen in Figure 2.9, certain trajectories take significantly more time to solve than other trajectories. The last potential factor is that newer versions of MATLAB can automatically run parts of the code in parallel using its intrinsic multi-threading [67]. This can result in the utilization of more cores than what the user specifies in the parallel toolbox. Method 2 is used to further investigate these potential causes for discrepancy between the optimal speedup and the observed speedup. This method, while not as user-friendly as method 1, has more options and tunable parameters that allow for further isolation and analysis of these potential inefficiencies.

Figure 2.12 illustrates the speedup curve that has been generated using method 2 to map

out the same footprint. Method 2 uses MPI to run multiple processors each running a separate instance of MATLAB without using the parallel toolbox. In addition, MATLAB's intrinsic parallelization has been disabled. This should remove any effects of unanticipated communication and built-in multi-threading that could be occurring when using the first parallel program architecture described above. Losses associated with coordination required to ensure memory coherency on a shared memory 12 core processor are still present. In addition, this method initially uses a work distribution method that is based on the number of points to be solved which is prone to the same issues of nonuniform work distribution among processors. A comparison between Figures 2.11 and 2.12 illustrates that the curves achieved using method 2 are fairly similar to that of method 1. This indicates that the primary causes of the differences between the optimal speedup and the actual has little to do with MATLAB's parallel toolbox or intrinsic multithreading. Further investigation is needed to distinguish how much the uneven work distribution and shared memory coordination factors contribute to the overall discrepancy between actual and optimal speedup.

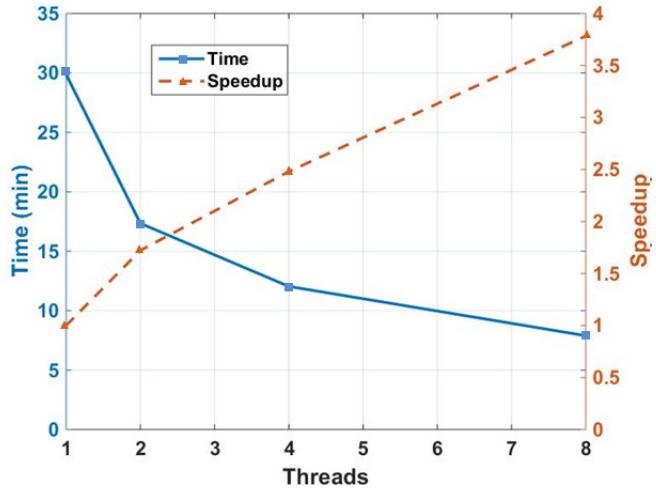


Figure 2.12. Speedup results for method 2 - shared memory (48 point footprint)

It is likely that one of the primary factors contributing to the difference between optimal speedup values and observed speedup values is a non-uniformity in the time required to solve each trajectory around the footprint. As the reader may recall, both method 1 and method 2 distribute the work among processors based on the number of trajectories that each

core should optimize. As an example, an 8 core case would distribute a 48 point footprint in a way that each processor will need to solve 6 trajectories. Figure 2.13 illustrates a stacked bar graph for the 8 core case using the current work distribution method where each core gets an equal number of points to find. It can be seen that this is not the optimal work distribution in terms of computation time among cores with thread 6 doing the most work and thread 3 doing the least.

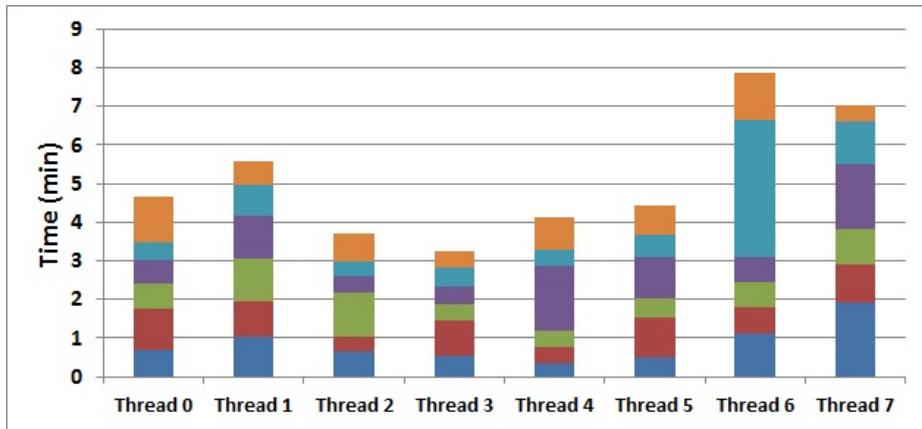


Figure 2.13. Work distribution for solving a footprint using 8 cores

In order to further understand how this workload balance is impacting speedup, the work is evenly distributed in terms of the computational time required for each core. This is not realistic in application, given that the knowledge of computation time for each point is used to accomplish this. However, it is a useful exercise that will help further isolate factors leading to inefficiencies in the parallel algorithm. Figures 2.14-2.16 illustrate a redistributed, time-balanced workload. They also compare the balanced workload on a simulated, distributed memory CPU (only utilizing a single core) with the same work distribution run on a shared memory CPU using multiple cores. The summary of these differences can be seen in Figure 2.17. This plot illustrates that there is a fairly significant slowdown resulting from the synchronization required between cores to ensure memory coherency on shared memory architectures. It can also be seen that this memory coherency cost quickly increases as the number of shared memory cores is increased. In fact, it can quickly be inferred that there is an ideal number of shared memory cores to use on a given problem for this reason. After this ideal point is passed, the overall computation time will begin to increase as more cores are added. The ideal number of processors is not only

a function of cores on a shared memory architecture, but problem size as well. Figures 2.14-2.17 illustrate the difference between a perfectly distributed memory architecture and a single multicore, shared memory CPU. The average shared memory penalty in these figures is determined by finding the average amount of time increase between solving the same set of trajectories on the simulated distributed case using a single core and the shared memory case using multiple cores on the same CPU. These two cases of distributed and shared memory represent two opposite ends of a spectrum in terms of hardware design. A more realistic approach that would fall in the middle of this spectrum would be to have several multicore CPUs running with a low number of cores on each to minimize the impact of memory synchronization. This would allow for efficiency by operating toward the left side of the memory coherency cost curve illustrated in Figure 2.17 without requiring such a large number of distributed memory CPUs. This would be a more practical approach that may be realized on future RLV systems. In addition, there are multicore CPUs that employ different memory architectures that can mitigate this issue [56].

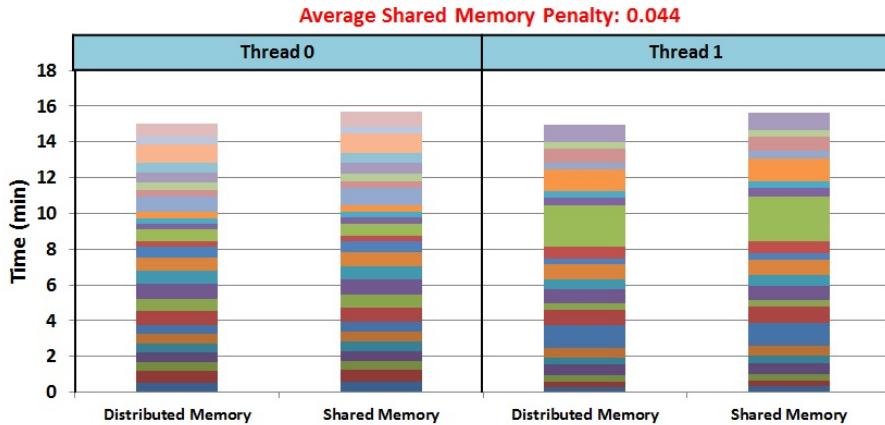


Figure 2.14. Time-based work distribution for solving a footprint using 2 cores

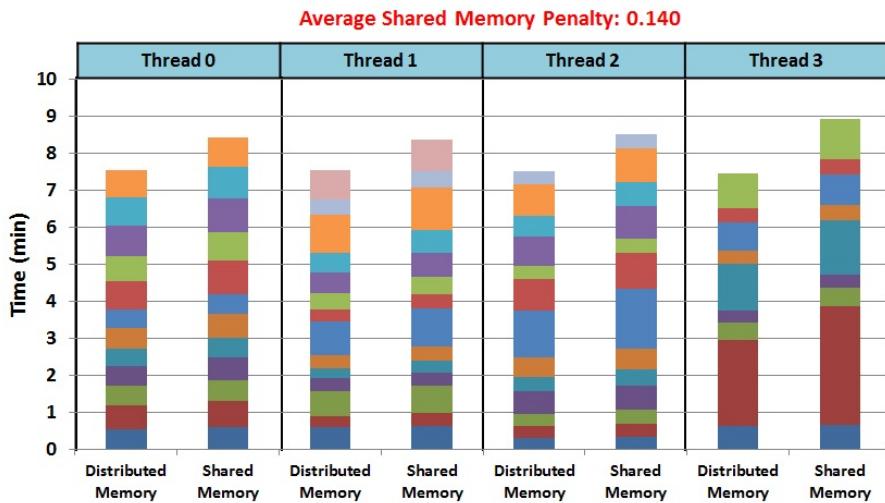


Figure 2.15. Time-based work distribution for solving a footprint using 4 cores

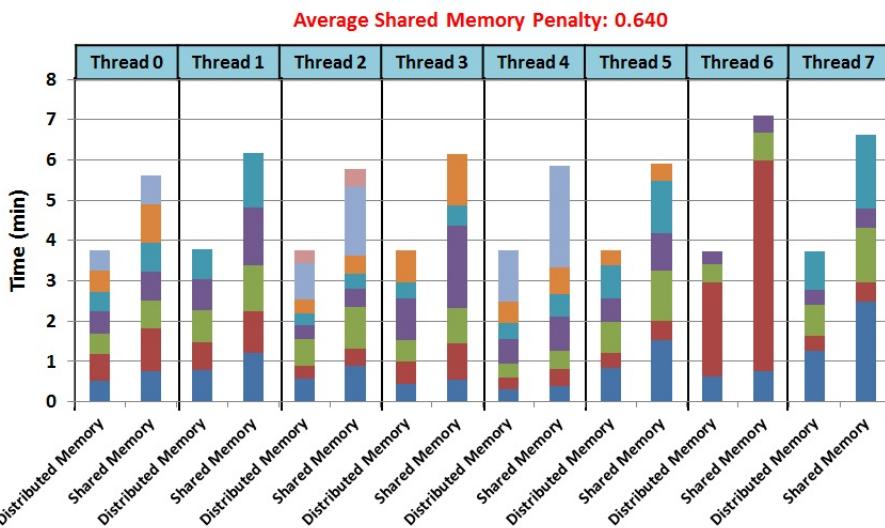


Figure 2.16. Time-based work distribution for solving a footprint using 8 cores

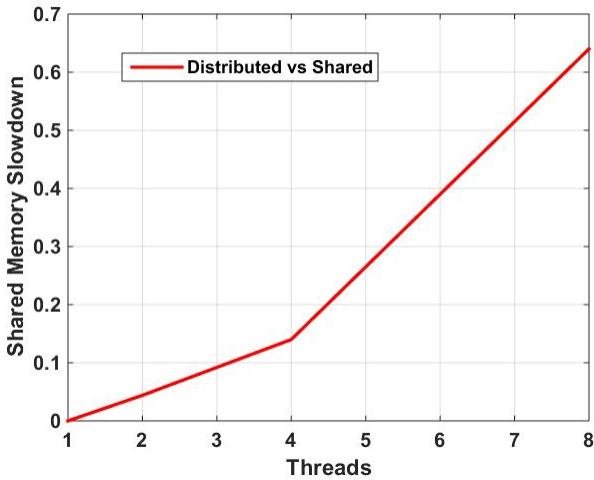


Figure 2.17. Shared memory coherency penalty curve

Finally, a comparison of time and speedup curves between four different cases are compared in Figure 2.18. It can be observed that the case with both a balanced workload and a simulated, distributed memory architecture approaches a near linear speedup curve which further validates the hypothesis that workload imbalance and shared memory coordination are the two primary causes of discrepancy between actual and linear speedup curves. By comparing the differences between both shared memory cases or both distributed memory cases, the effects of work imbalance can be observed. On the other hand, by comparing differences in both balanced cases or both imbalanced cases, the effects of shared memory coordination can be observed. It quickly becomes apparent that work balance is more important with lower numbers of cores, however, memory coordination starts to outweigh work imbalance issues as the number of processors is increased. This point is further illustrated by looking at where the time curves of the (Shared/Balanced) case and the (Distributed/Unbalanced) case cross. This crossover point is where the importance of distributed memory overtakes the importance of workload balancing.

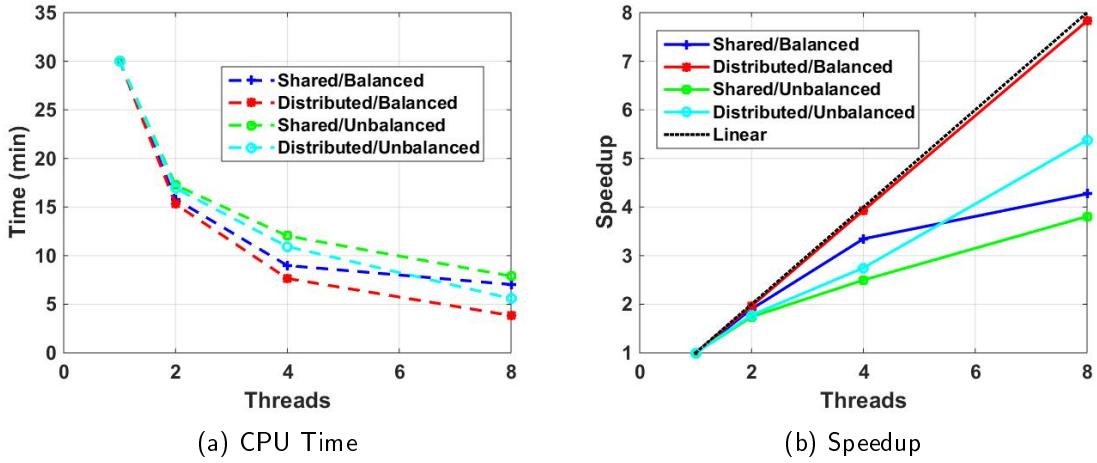


Figure 2.18. Speedup comparison for method 2 comparing workload balancing and memory models

2.9 Conclusion

Two different parallel program architectures have been applied to solve an optimal ground footprint for a RLV problem. The results show that speedups of approximately 4 can be achieved with both methods as the number of processors is increased to 8 on a shared memory multi-core CPU without any sophisticated work balancing scheme. It is determined that the non-uniform distribution of required computational time for each trajectory and the coordination required for coherent read/write memory access in shared memory CPUs are the two primary causes for differences between optimal linear speedup and the actual speedup observed for each method. MATLAB's parallel toolbox and intrinsic multi-threading have little to no effect in terms of impacting parallel program performance for this specific problem. The memory coordination issues can be addressed by running the algorithms on hardware with distributed memory architectures, in other words, computers or compute nodes that have multiple processors with separate memory caches. Using this type of computing device, it is theoretically possible to reduce the calculation time required to generate an entire RLV footprint down to the time required for the longest serial computation. In this case, the longest computation time required to solve a single point on the footprint. Achieving this optimal speedup would require the same number of processors as trajectories, or points on the perimeter of the footprint. In addition, a more-sophisticated

work distribution algorithm that splits up the work evenly in terms of computational time required per processor would help enhance the efficiency of these algorithms when there are not as many cores as points on the footprint. While MATLAB is well-suited for demonstration purposes, additional speedup on operational versions of these algorithms would be obtained by utilizing higher performance computing languages such as C, C++, or Fortran.

It can be seen that this methodology and way of thinking can be applied to a number of similar engineering problems. This chapter presents just one example of where the fairly recent multi-core revolution in CPU development can be used as an enabler to more efficient ways to solve problems such as the landing footprint of a RLV. This chapter simply took advantage of the “perfectly parallel” aspect of footprint generation. However, each of the trajectory optimization tasks themselves could be parallelized for further efficiency and speedup. This of course would be a more difficult problem due to the fact that an increased level of communication and synchronization would be required among the processors involved. The remainder of this dissertation investigates and develops various evolutionary algorithms that could potentially be applied to solving and optimizing the individual trajectories in parallel. The parallel computing concepts and lessons associated with factors that impact the efficiency of parallel computation will be continue to be leveraged throughout this dissertation. It will be seen that parallel computation proves to be a valuable tool in addressing several of the key challenges associated with evolutionary algorithms.

CHAPTER 3: Genetic Algorithms

3.1 Introduction

Now that various ideas of parallel computation and application examples have been discussed, this chapter shifts the focus of this dissertation toward the investigation of evolutionary algorithms. These types of algorithms are a set of optimization techniques that lend themselves nicely to parallelization given that they operate on sets of candidate solutions and utilize a number of independent operations. The GA is investigated first, as it is the most popular of the various forms of evolutionary algorithms. GAs were introduced by John Holland in 1975, as heuristic search algorithms based on biological evolutionary processes [39]. They were originally applied to solve combinatorial problems with discrete decision variables. One of Holland's students, Ken De Jong, was the first to suggest the application of GAs as function optimizers. In De Jong's Ph.D. dissertation, continuous optimization problems were converted to combinatorial problems through binary encoding and solved using GAs [68]. Even though GAs were first introduced in 1975, it is interesting to note that the first documented GAs that utilized real-coded variables for optimizing continuous functions did not begin to appear until 1989 with Davis adapting GA operator probabilities in real-time and with Lucasius et al. solving chemometrics problems [69], [70]. Over the years, GAs have become very popular due to their simplicity, their direct parallels to familiar concepts in biology, and their ability to be applied as black box function optimizers. Black box function optimization is needed when knowledge of the specific model or function to be optimized is not known. This can be the case with complicated models, real-world systems such as factories and assembly lines, or when restrictions on information transfer (e.g. proprietary data) limit the ability to obtain gradient information that is often required for conventional optimization schemes. In addition, GAs have no requirements for continuous or differentiable search spaces, making them ideal for complex problems with discontinuous search domains. In practice, GAs have been applied to a wide range of applications to include: interplanetary mission design [11], calculating commercial aircraft approach trajectories [71], optimizing networks of pipes for water delivery [72],

and have even been used by musicians to generate jazz solos [2].

This chapter provides the reader with the necessary background and understanding for the following chapter in which several types of parallel GAs are developed and applied to an optimal control problem. It begins with a description of the various GA operators and standard implementations for binary encoded GAs. Next, the chapter presents a top-level overview of the original and modern theoretical approaches to understanding the dynamics and convergence properties of these algorithms. Next, the theoretical and empirical approaches to tuning the user-defined parameters in these algorithms are developed and presented in detail. After this, the chapter introduces the reader to a number of improvements and recent concepts used to assist these algorithms in solving different types of challenging problems. It then transitions toward the development and discussion of RCGAs and their associated operators. From here, a top-level theoretical overview of the dynamics and convergence properties of RCGAs is developed. Lastly, the chapter concludes with an overview of the various forms of parallel GAs, their associated advantages and drawbacks, and a brief theoretical analysis of each parallel GA type based on concepts from literature.

In a GA, Darwinian concepts are applied to an initial “population” or set of candidate solution vectors to a given problem. After this initial population has been generated, each individual candidate solution or “chromosome” is evaluated based on its “fitness,” or objective function value. The chromosomes then compete against one another in a survival of the fittest scenario where the most fit chromosomes have the highest probability of surviving. These surviving chromosomes become parents that then generate “offspring” through a simulated reproduction process referred to as crossover to provide the next “generation” of solution vectors for the algorithm. As in nature, a random search factor is added to this process in the form of genetic “mutation.” A summary of the steps in a standard GA is illustrated in Figure 3.1. Each of these steps is described in greater detail throughout the following sections of this chapter.

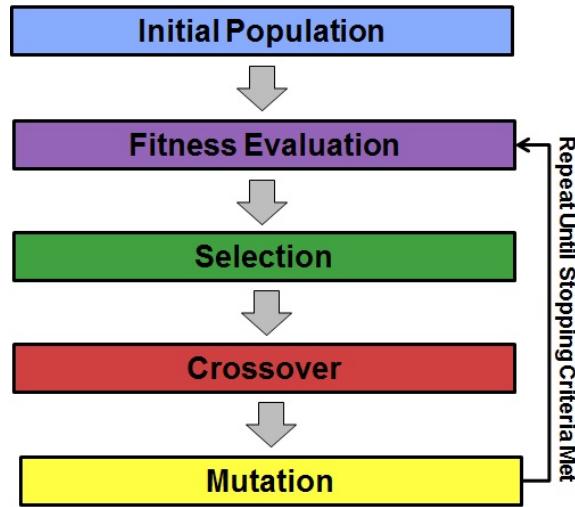


Figure 3.1. An illustration of a simple GA

In order to help the reader understand the logic and reason behind each step in the genetic algorithm, the concept will initially be described as a simplistic optimization algorithm that is progressively improved until the final product is a somewhat sophisticated GA. In the most fundamental sense, an iterative optimization algorithm is one that repeats an operation, or series of operations in order to try to iteratively approach the solution to a given problem. To start with the most basic form of optimization algorithm, one could imagine a simple search algorithm that randomly generates a candidate solution to an optimization problem during each iteration. With this approach, a few things quickly become necessary for this algorithm to function properly. First of all, the algorithm needs some way of determining how good a given candidate solution is. In the realm of GAs, this metric is often referred to as the ‘fitness’ of a candidate solution or ‘chromosome’ as it is referred to in much of the literature on GAs.

3.2 Fitness Function

The fitness function is the only link between the GA and the problem that it is trying to solve. The simplest fitness function is the objective function of an unconstrained optimization problem as described in Equation (3.1) where \bar{x} is a solution vector and n is the problem dimension. In a maximization problem, the candidate solution or chromosome that has the highest fitness value would be considered the best solution found. Of course, the world

of optimization is seldom this simple and their are typically large numbers of inequality and equality constraints that can be highly nonlinear. However, for the purposes of this chapter, it is easiest to assume that the initial problem is unconstrained, and the fitness function is exactly representative of the initial optimization problem. In other words, it is assumed that the fitness function is the objective function. In fact, unconstrained NLPs are the only form of NLPs that a standard GA is able to solve without the introduction of any constraint handling techniques. Appendix B provides a survey of techniques that can be used to handle constrained optimization problems in GAs. Furthermore, several types of constraint-handling techniques will be introduced and analyzed in Chapter 4. However, for this chapter, it is best to assume the GA is attempting to solve an unconstrained NLP problem.

$$\begin{aligned} & \text{minimize } f(\bar{x}) \\ & \text{s.t. } \bar{x} \in \mathbb{R}^n \end{aligned} \tag{3.1}$$

Going back to the simple optimization algorithm, the random search algorithm is now able to evaluate each of its randomly generated candidate solutions. Next, it is useful for this algorithm to have some type of memory, where it keeps track of the best solution found over all of the iterations. It has been shown that such an algorithm is guaranteed to converge as the number of iterations approaches infinity [73]. While this is an optimization algorithm that has been proven to converge, engineers typically don't have infinite amounts of time to wait for an optimal solution. This idea is further developed in Section 3.12.1 of this chapter. Instead of creating a single candidate solution at each iteration, a GA works on a set or ‘population’ of candidate solutions at each iteration for better exploration and exploitation of the search space. Figure 3.2 further illustrates this comparison between a GA that operates on multiple solutions from one iteration, i , to the next and a standard deterministic algorithm that iterates on a single point in the search domain. In addition to improving algorithm search capability and speed, this idea of operating on populations of solution vectors makes these algorithms very amenable to parallel processing as will be seen later.

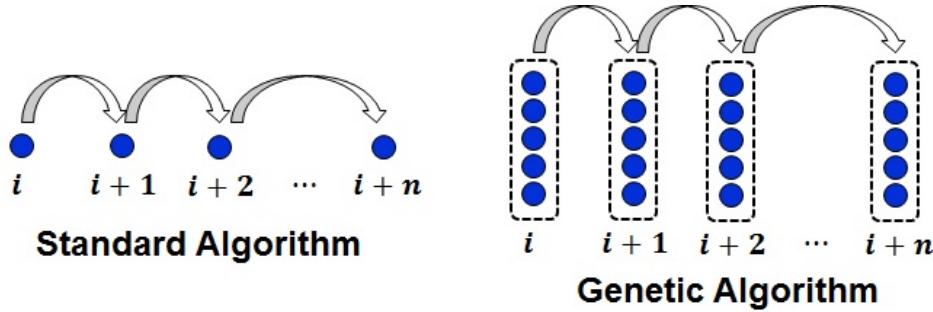


Figure 3.2. A comparison of a standard algorithm and a GA

3.3 Initial Population

An initial population or set of solution vectors is typically created using a uniform, random distribution that samples the entire search space choosing an initial set of trial solutions or chromosomes. This is common practice given that GAs are typically used on problems where there is little to no *a priori* knowledge about the problem in regard to where its optimal solution is likely to be [73]. However, there are different ways to construct an initial population distribution to leverage problem-specific knowledge or promote greater initial population diversity (e.g. initial sizing and grouping of structural members for truss designs based on compression force, tension force, or zero-force loads) [74].

Next, assume that the search algorithm is further modified to reflect a survival of the fittest technique where it enforces a simulated natural selection or pruning process that occurs from population to population. In other words, some type of fitness-based selection occurs to force the future generations to have better average fitness than previous generations. Before some of the potential issues associated with this new algorithm are pointed out and addressed, it is first important to understand some of the different fitness-based selection methods that are typically implemented in GAs.

3.4 Selection Operators

The selection process is very important to GA performance in that it is what creates the “survival of the fittest” attribute that forces populations to improve over time. Selection techniques often leverage the values from the fitness function evaluated at each candidate

solution to refine the current population in a way that leads to a higher average fitness among parents that have been chosen to produce the next population. There are several different selection methods [75], [76] that are described in more detail in the following subsections. Bickle et al. have conducted a number of studies and experiments to model and compare the different selection techniques outlined in the subsections below [76]. After comparing the various selection schemes and looking at traits such as: selection pressure, loss of population diversity, and variance of the resulting population, they conclude that the best selection operator is problem dependent [76].

3.4.1 Roulette Selection

Roulette selection is a simple selection process where each member of the population has a probability of selection that is proportional to their fitness value. This class of selection techniques is often referred to as proportionate selection [75]. One of the simplest implementations is to start with a given population and evaluate all of the associated fitness values for each member. From here, the algorithm sums up all of the fitness values to get the total fitness for the population. This algorithm assumes that all of the fitness values have been scaled in such a way that they are all positive. This process is more intuitive for a maximization problem where the largest fitness has a proportionally large percentage of being selected, however, the fitness values can be adjusted and scaled so that minimization can be done using this technique as well (e.g. $\frac{1}{fitness + \epsilon}$). Next, a random fitness value is chosen between zero and the sum of all fitness values in the population. This value is treated as the threshold value. With this value set, the algorithm simply starts randomly picking chromosomes and summing up the cumulative fitness value as they are selected. The randomly chosen chromosome that makes the cumulative fitness go above the threshold value is chosen to be a parent for the next population. As can be imagined, the chromosomes with the larger fitness values have a greater chance of being the chromosome that causes the cumulative fitness sum to break the randomly chosen threshold value. This entire process is repeated N_{pop} times until enough parents have been generated to create a new population of N_{pop} individuals through crossover and mutation (described later). This process produces a similar result to that of a roulette wheel where each chromosome has a fitness proportional slice of area as depicted in Figure 3.3. This wheel is then spun N_{pop} number of times to create the next generation of parents that will undergo the simulated reproduction process of crossover and mutation.

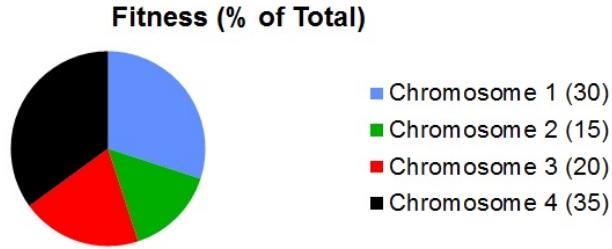


Figure 3.3. An illustration of a roulette selection operator

3.4.2 Tournament Selection

Tournament selection is a fairly common and straightforward selection method. As illustrated in Figure 3.4, a user-defined number of chromosomes or tournament pool is chosen at random from the current population. The chromosomes in this tournament pool compete with each other and the one with the best fitness value is selected to be a parent for production of the next generation (minimization is used in this figure as an example). This process is repeated, another pool is randomly selected and a winner is determined until a user-defined population size N_{pop} number of parents have been selected to produce the next generation. There are various forms of the standard tournament selection scheme. Some GAs utilize a tournament selection operator that only considers the current population of offspring while other GAs allow the selection operator to consider both the previous generation of parents and their offspring. In addition there is the option of replacement vs. no replacement when the tournament selection operator is selecting from both the offspring and the previous set of parents given that it is selecting from a set of solution vectors that is twice the size of the population. Replacement allows the selection algorithm to potentially pick the same chromosome multiple times, while not using replacement forces the selection operator to only choose from chromosomes that have not yet been selected. It must be noted that replacement is required when only considering the current population of offspring solution vectors and not the parents. This creates a selection operator where the solution vectors with the best fitness will likely be represented multiple times in the selected set of parent chromosomes.

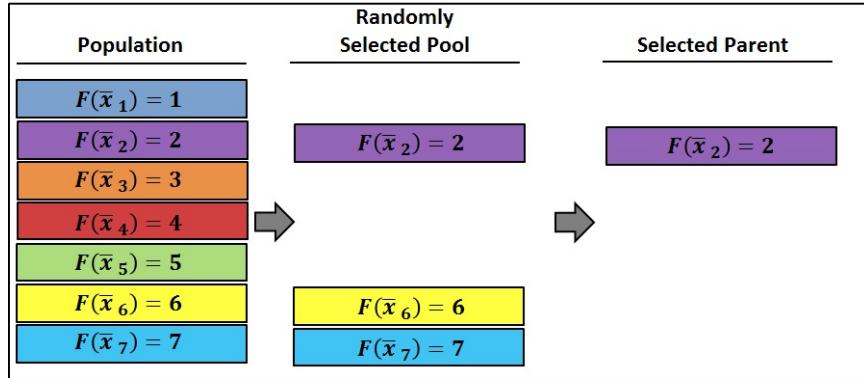


Figure 3.4. A diagram of a tournament selection operator (minimization)

3.4.3 Ranking Selection

There are a number of various GAs that have been developed utilizing a rank-based sorting approach [77], [78]. In these algorithms, a sorting algorithm is employed to sort the population based on each member's fitness values. After this is completed, rank-based weights are assigned to each chromosome in a consistent fashion from generation to generation so that the rank of each chromosome defines how likely it is to be selected to be a parent for the next generation. After one of several rank-based weighting functions is applied to the sorted population, simple roulette or proportional selection is conducted. There are a number of different linear and exponential rank-based weighting schemes that can be applied [76]. This method allows for a consistent selection pressure from generation to generation regardless of the relative distribution of fitness values. This can be useful when the population is very similar in terms of fitness values. This type of selection allows the GA to continue progression toward the better solutions, where a standard roulette technique would only slightly favor the best solution in the case where fitness values are relatively close. While this seems like a superior method, these sorting algorithms can become computationally expensive when compared to the previous algorithms as the size of the population grows.

3.4.4 Truncation Selection

Truncation selection is a straightforward technique that is more common in the evolution strategy world than it is in the realm of GAs [79]. Truncation selection also utilizes a sorting technique to gain knowledge of the rank order of a given population. From this point, the

truncation selection operator truncates the population by only looking at a set number of the highest performing chromosomes. From this subset, a simple random selection technique can be implemented where each member of this higher performing subset has an equal chance of being selected [80]. Again, this method has the drawback of additional computational complexity due to the requirement of sorting the population based on fitness during each generation.

3.4.5 Elitism

Elitism is the simple method of retaining the best member, or several members from the current population and ensuring that they remain in the next population [73]. This is a very useful practice that significantly increases GA performance for any of the above selection techniques without much additional computational work. It ensures that the most fit member(s) are not lost in the process of selection, crossover, and mutation. This technique helps the rest of the population increase their fitness quickly which in turn searches areas in the neighborhood of the best solution found. In fact, it is seen later in Section 3.12.1 that convergence can't be proven for non-elitist GAs, i.e. those without memory of the best solution found. However, it can be shown that a simple memory that retains the best solution found to date, while not an elitist GA, does still converge given that the optimal solution is eventually discovered as the number of generations approaches infinity [81].

3.5 Encoding

Standard GAs encode the decision variables of the problem in a way that lends itself to the genetic operators described in the Section 3.6. The standard, most simplistic GAs often use some type of binary encoding. Binary encoding has a cardinality of 2, or in other words, each bit can have 2 potential values, either a 1 or a 0. Each bit location represents a specific power of 2. All of the bit locations, or particular powers of 2 that have a 1 in them are summed together to produce a single integer. The rightmost bit location in a binary string of length l represents 2^0 , and from left to right the location representation goes as follows: $2^{l-1}, 2^{l-2}, \dots, 2^1, 2^0$. As an example, the binary string $[1, 0, 0, 1] \rightarrow value = 2^3 + 2^0 = 9$. It can quickly be inferred, that the maximum number that a binary string of length l can represent is $value_{max} = 2^l - 1$. If the user has a range or upper and lower bounds that they would like to enforce on a decision variable, these binary numbers and resultant integers

can easily be scaled so that they represent a floating point number using Equation (3.2).

$$variable_{scaled} = \frac{value}{value_{max}}(UB - LB) + LB \quad (3.2)$$

It is fairly intuitive to see that the resolution on the scaled variables directly depends on the number of bits used in the original binary string combined with the size of the range or distance between the upper and lower bounds. This relationship is defined in Equation (3.3) where UB is the upper bound, and LB is the lower bound.

$$resolution = \frac{UB - LB}{value_{max}} = \frac{UB - LB}{2^l - 1} \quad (3.3)$$

Often times in GA literature, the encoded variables are referred to as a “genotype” and the decoded variables are referred to as a “phenotype.” [73] This is in line with the biological roots of GA concepts.

3.5.1 Binary Coding vs. Reflected Binary Gray Coding

In standard binary representation, there is always the possibility that a change in a single bit (a distance of 1 in Hamming space) could lead to a significant change in the decoded value of the given variable. This idea is referred to as a Hamming Cliff [82], and is typically viewed as an unwanted characteristic of binary encoding due to the fact that it can force the GA to behave in unanticipated ways. It is important to note that the GA utilizes structure in the encoded variables. The algorithm uses the idea that genotypes which are relatively close in Hamming space should be similar in fitness or performance. A Hamming Cliff illustrates an instance where this is no longer the case, making it difficult for the GA to use the encoded structure to arrive at a near-optimal value due to this disconnect between genotype and phenotype. An extreme case of this would be if one were to use a completely random encoding scheme that changes every time a chromosome is encoded. Even though the GA would be trying to utilize crossover as a way to take advantage of the search space structure, it would ultimately be no different than a random search [82].

Reflective Binary Gray coding is one of the more common encoding techniques that is used to avoid Hamming cliffs by more closely relating the encoded problem space to the real problem space. Gray coding is a way to ensure that adjacent numbers in Euclidean space are

also adjacent in Hamming space [82]. As can be seen in Equation 3.4, it is fairly simple to convert from standard binary encoding to gray code and back to binary again. In the below equation \oplus is a bit-wise exclusive OR operator, i.e. the sum of the bits being compared modulo 2. In addition, x_{bi} is a binary string of bits and x_{gr} is the same string in Reflective Binary Gray coding.

$$\begin{aligned} \text{Binary to Gray} &:= \begin{cases} x_{gr}(1) = x_{bi}(1) & \text{if } i = 1 \\ x_{gr}(i) = x_{gr}(i - 1) \oplus x_{bi}(i) & \text{if } i > 1 \end{cases} \\ \text{Gray to Binary} &:= \begin{cases} x_{bi}(1) = x_{gr}(1) & \text{if } i = 1 \\ x_{bi}(i) = x_{gr}(i - 1) \oplus x_{gr}(i) & \text{if } i > 1 \end{cases} \end{aligned} \quad (3.4)$$

Table 3.1, recreated from Caruana's paper [82], illustrates the use of the conversion equations and how the Hamming distances compare between the two encoding types. It can be seen that standard binary encoding can have two adjacent numbers in Euclidean space that have a Hamming distance that is up to the length of the bit string itself e.g. the Hamming distance between seven and eight is four. However, it can also be seen that Reflective Binary Gray encoding ensures that all adjacent numbers are only different by a Hamming distance of 1.

Integer	Binary	Hamming Distance (n,n-1)	Reflective Binary Gray	Hamming Distance (n,n-1)
0	0000	—	0000	—
1	0001	1	0001	1
2	0010	2	0011	1
3	0011	1	0010	1
4	0100	3	0110	1
5	0101	1	0111	1
6	0110	2	0101	1
7	0111	1	0100	1
8	1000	4	1100	1

Table 3.1. A comparison of Hamming distances in binary and reflective binary gray encodings

Caruana and Schafer study the differences between encoding types in more detail using empirical results from a simple GA applied to six test problems with both types of encodings to determine that the Reflective Binary Gray encoding outperforms the standard binary encoding with statistical significance on two problems and performs equally as well on the other four [82]. They conclude that in general, gray encoding seems to outperform standard

binary coding in GA on problems with parameter values that are ordered quantities. Whitley illustrates that while, for most problems gray encoding induces a fewer number of local optima and results in simpler encoded problem spaces and superior performance when compared to binary, there are some cases where the opposite is true and simple binary encoding performs better [83]. His study indicates that, for the majority of problem types, gray coding performs better and essentially has a “free lunch” advantage over the standard binary encoding.

3.6 Crossover

Now that some common selection methods and encoding implementations have been introduced, a return to the simple optimization algorithm is warranted. It can quickly be pointed out that this algorithm will eventually converge to a uniform set of candidate solutions. Based on the initial population, the most fit ‘chromosomes’ are favored and will likely be represented multiple times in the next generation. As this occurs over and over again, the populations will quickly become entirely composed of a number of copies of the same candidate solution. If the initial population is infinitely large, this might not be a major concern, however, with finite populations, there is no guarantee that the initial population contains a globally optimal solution, or even a solution that is ‘good enough’. With this, the best that such an algorithm can do is converge to a population that is filled with copies of the best solution originally obtained in the initial population. In order to overcome this shortfall, GAs utilize simulated reproduction techniques referred to as crossover operators. There are several types of binary crossover techniques that have been studied in detail [84], [85]. It must be noted that these crossover operators are fundamental to the GA’s ability to efficiently evolve populations toward optimal points.

3.6.1 Single-Point Crossover

The original version of crossover is the simple single-point method utilized in John Holland’s original GA [39]. In this technique, the selected parent population is paired off by twos to create $\frac{N_{pop}}{2}$ sets of parents. Each set of parents then produces a set of two offspring as a result of the crossover process. The chance that crossover will occur, p_{cross} , is another user-defined parameter and is typically on the higher side ($p_{cross} = [0.5 - 1.0]$) [86]. If the algorithm decides that crossover should occur between the two parent chromosomes,

a uniformly random bit location is chosen as the pivot point or crossover point. At this location the genetic information to the left (or right) of the point is swapped between the two parent chromosomes to produce two offspring chromosomes. If the algorithm decides crossover should not occur for a given set of parents, then they remain unchanged and become offspring for the next generation. This standard crossover operator is illustrated in Figure 3.5.

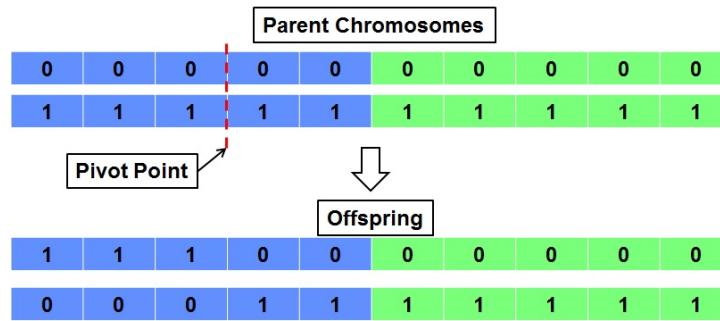


Figure 3.5. A single-point crossover operator

3.6.2 Multi-Point Crossover

Multi-point crossover works in a similar way, only with more than one crossover point. This increases the amount of disruption of the original parent vectors, and has been shown to increase performance with some applications [84]. After the crossover points have been determined, the genetic material is swapped in a method similar to that depicted in Figure 3.6. The genetic material between two selected points that is to the right of an even (or odd) number of pivot points is swapped to produce two unique offspring.

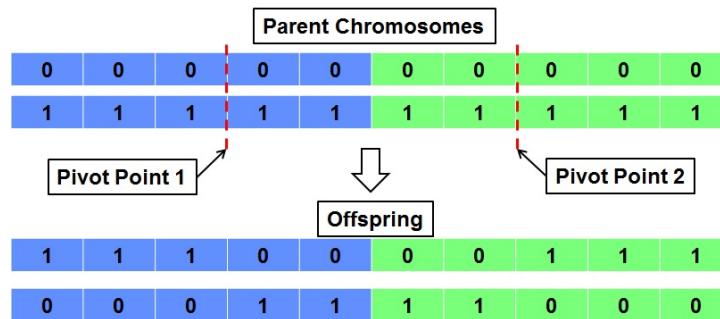


Figure 3.6. A multi-point crossover operator

3.6.3 Uniform Crossover

Uniform crossover looks at one gene, or bit, at a time and has a 50% chance of choosing the gene from the first parent and a 50% chance of choosing the gene from the second parent [84]. This process can be further understood by looking at Figure 3.7. It can be seen that a mask string is randomly produced, where a 1 indicates that that specific gene location is swapped between parents and a 0 indicates that each parent retains that specific gene.

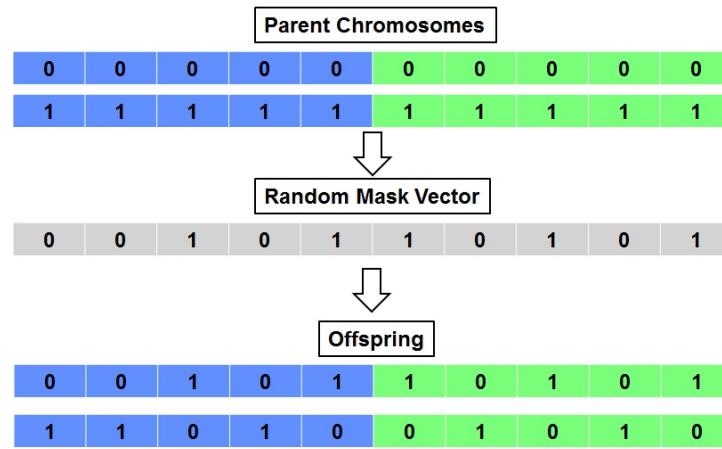


Figure 3.7. A uniform crossover operator

With all of the above crossover operators available for use, which one is the best? According to Holland and the Schema Theory (discussed later), crossover operators that minimize disruption and preserve critical building blocks should be superior. However, several authors have shown that multi-point and uniform crossover seems to outperform single point crossover on a number of test problems [84]. De Jong and Spears look further into this, and determine that these crossover operators have to be considered with population size [85]. They indicate that the uniform and multi-point crossover operators perform better only in cases where the population isn't sufficiently large to fully sample the problem space. In these specific instances, increased disruption caused by more crossover points helps increase the exploration property with smaller populations that quickly become homogeneous and thus create difficulty for single point crossover to produce new individuals. However, when sufficient population sizes are used, the lower disruption crossover techniques actually outperform the uniform crossover technique [85]. This provides another theoretical and empirical data point that indicates population size is not only problem dependent, but also

algorithm dependent. Population sizing will be addressed later in this chapter.

3.7 Mutation

Standard optimization algorithms along with GAs (as described up to this point) have the potential shortfall of converging to a sub-optimal point or a local minimum. This becomes increasingly likely for problems that are highly multimodal. GAs draw on genetic mutation, a phenomenon that naturally occurs in biology, to allow the algorithm to escape local optima [39], [68]. This process typically occurs after one of the above crossover techniques has been applied to the chosen parent chromosomes to create a new population of offspring chromosomes. Figure 3.8 illustrates a simple bitwise mutation process that is commonly used in binary GAs. The algorithm steps through each bit of each offspring chromosome with a user defined probability of mutating or flipping each bit. This mutation probability p_{mut} is typically small ($p_{mut} = [0.001 - 0.05]$) so as to not disrupt the evolution process [86]. If this percentage is too high, the algorithm begins to behave as a population-based random search where the evolution of the algorithm is dominated by the random mutation operator. Later sections on GA theory will illustrate how the GA is more efficient than a random search when the above operators are allowed to operate effectively without too much disruption from the mutation operator.

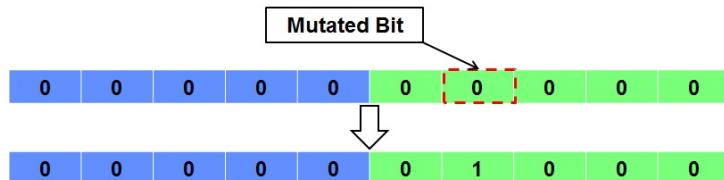


Figure 3.8. A bit-wise mutation operator

3.8 Stopping Criteria

GAs are iterative algorithms that require some type of stopping criteria to prevent them from going into infinite loops. There have been a number of proposed stopping criteria, all

of which have their place for given problems and applications. In fact, multiple stopping criteria are often used in conjunction. As will be seen in the following sections, a practical and robust globally optimal convergence proof does not exist. Given this, GAs are often designed to work on the practical basis of finding solutions that are sufficiently good for the practitioner and the associated application. With this in mind, the simplest stopping criteria are simply user-defined maximum number of generations or maximum amount of computational time. As soon as this generation or computation time threshold is met, the GA stops iterating and returns its best solution found. One of the more reasonable stopping criteria is to have a target objective function value that the GA is trying to achieve. As a stopping criteria, once the GA finds a solution vector that meets this target value, the algorithm stops the iteration and returns the solution. As can be imagined, it may be useful to combine this stopping criteria with a generation or computation time threshold so that an infinite loop is not encountered when an overly optimistic fitness target is used.

Another stopping criteria that is fairly common is to estimate when the algorithm has converged to a solution [73]. This is often done by looking at the population average fitness value from generation to generation. A user-defined tolerance can be set as the minimum amount of difference between the average fitness values of the current population and the preceding one. Along these same lines, the standard deviation of population fitness values can be utilized as another potential convergence metric. Where a very small standard deviation indicates that all of the chromosomes in a population are close to identical and the GA has converged to a potential solution.

More recently, several researchers have been investigating increasingly sophisticated stopping criteria where Karush-Kuhn-Tucker (KKT) conditions are estimated and applied to develop a measurement of proximity to a locally optimal point [87]–[89]. While these techniques offer a more robust stopping criteria for certain problems, they do require the use of local gradient information of the search space. This idea seems to go against one of the primary advantages of GAs in that they are able to treat the problem as a black box without the use of gradient information or the need for continuous, differentiable search domains. These authors have developed methods to deal with discontinuous search spaces, but still require more information from the problem than standard GAs require. This may be a practical stopping criteria for some applications, but might not be feasible when little information is known about the problem at hand and a true black box optimization method

is needed.

3.9 Schema Theory

When studying GAs, it is important to introduce John Holland's Schema Theory [39] as much of the early theory and fundamental understanding of how GAs work up through the early 1990's is directly based on these concepts. It must be noted that this theory has been criticized and brought into question by several authors over the years; however, many of the concepts and fundamental ideas remain at the core of how GAs are thought to operate [73]. The idea of a schema is analogous to a hyperplane in geometry, or simply a subspace in a given encoding with a defined cardinality or alphabet. As a simple example, assume an encoding with a cardinality of two (binary encoding). A schema represents a certain subspace of all the possible bit strings of a given length. As an example, consider a 4 bit string with the “*” symbol representing a free bit or wildcard bit that could be either 1 or 0. A given schema could be represented as (**11). Using this definition, the following bit strings are all members of this given schema: (1111), (0011), (0111), (1011) as they all possess the defining bits (11) in the 3rd and 4th position. Looking at this through the lens of a GA, each of the previously mentioned schemata (plural for schema) have an assigned fitness value. Holland will define the schema fitness for (**11) as being the average fitness value of all strings that fit within this defined subset of strings or hyperplane in a given population. The fundamental idea is that the GA tends toward schema that have the best schema fitness values by solving many multi-armed bandit problems [73]. A two-armed bandit problem is one that has received a great deal of attention and study in the statistics world. It can be thought of as a problem where two events have different unknown outcome distributions and the task is to maximize the return over a given number of trials. The only way to generate an understanding of which event has a superior outcome distribution is to explore by running trials for each event and measuring the outcome. However, at a certain point, the user would like to capitalize or exploit the knowledge that they have gained through exploration before all of the trials have been used. Holland applies this problem to the sampling and representation of various schema in the GA population throughout the run of the algorithm. He also uses the term implicit parallelism to describe the idea that the GA is solving a large number of multi-armed bandit problems simultaneously as it is applying its various genetic operators and deciphering which schemata have the best average fitness

based on sampling. [73]

It can be inferred that there are 2^l possible schemata that can exist within a single chromosome of length l . In a population, there could then be $N_{pop}2^l$ schemata, however, this doesn't account for duplication and overlapping schemata so it is a conservative upper bound on the number of schemata in a given population. With several assumptions, John Holland was able to derive that the lower bound of schemata being processed by a GA is on the order of $O(N_{pop}^3)$. However, this is one of the areas that is contested. Bertoni and Dorigo illustrate that this bound is somewhat arbitrary and is entirely dependent on the relationship between the populations size and the defining length of a given schema [90]. Holland assumes that the value of this ratio is equal to one for his derivation, but Bertoni and Dorigo illustrate that this assumption can take on other values and can significantly change this lower bound on the number of schema being processed by the GA.

The primary result of the Schema Theory is the derivation of an expected number of a given schema going from generation g to $g + 1$. Before this derivation is described, it is important to first introduce several key schema characteristics. The first is the schema order (O_s) which is the number of defining bits within a given schema, where a defining bit is one that is not represented by a wild card character. Next, the schema defining length, l_s , is the distance in bits between the leftmost defining bit and the rightmost defining bit. First, the effect of a standard fitness-proportional selection operator on the number of a given schema, N_s , in a population of size N_{pop} from the initial generation g to the next $g + 1$ is analyzed. Equations 3.5 and 3.6 illustrate the average schema fitness, f_s , and the average population fitness, \bar{f} . Equation 3.7 shows the expectation of the number of schema in the $g + 1$ generation as the result of fitness-based selection alone [73]. This will be pieced together with the effects of crossover and mutation to develop the full expectation for the number of a given schema going from one generation to the next.

$$f_s = \frac{\sum_{x:S \in population} f(x)}{N_s} \quad (3.5)$$

$$\bar{f} = \frac{\sum_x f(x)}{N_{pop}} \quad (3.6)$$

$$E_{selection}[N_s(g + 1)] = N_s(g) \frac{f_s}{\bar{f}} \quad (3.7)$$

From here, the effects on a given schema as a result of a standard single-point crossover operator are investigated. More specifically, the interest is in the chances of crossover disrupting a given schema that has made it through the selection operation. Recalling how single-point crossover works from the previous section, it is assumed that crossover will disrupt a given schema if the crossover point occurs between the leftmost and rightmost defining bit of the schema. In order to keep things simple, Holland assumed that all parents containing a given schema that experience a crossover point within their defining bits are disrupted. In addition, it is assumed that no new instances of the given schema are created during the crossover operation [73]. Vose points out that the assumption of no new schemata being created from crossover essentially means that only schema that are present in the initial population can be involved in the progression of the algorithm [44]. This unfortunately results in the neglect of a very important aspect of genetic search regarding the creation of new schema that shouldn't be overlooked. Equation 3.8 illustrates the probability that a given schema that will survive the crossover process. It can be seen that this crossover survival probability is a function of crossover probability, p_{cross} , schema defining length, l_s , chromosome length, l , and the probability that the two selected parents are different, p_{diff} .

$$1 - p_{cross} \frac{l_s}{l-1} p_{diff} \quad (3.8)$$

Equation 3.9 describes the chance that a given schema will survive a standard bitwise mutation process as a function of probability of mutation, p_{mut} and schema order, O_s .

$$1 - O_s p_{mut} \quad (3.9)$$

Lastly, all of this is combined to set a lower bound on the number of a given schema that makes it from generation g to generation $g + 1$. Equation 3.10 is simply a combination of the above effects of selection, crossover, and mutation. Given that a lower bound is desired, p_{diff} is set to 1 which is an assumption that all parents chosen for crossover are never the same. While this isn't always the case in practice, this will result in a maximum disruption from crossover which results in a lower bound for the expected number of a given schema to survive from one generation to the next.

$$E[N_s(g+1)] \geq \left(N_s(g) \frac{f_s}{\bar{f}} \right) \left(1 - p_{cross} \frac{l_s}{l-1} - O_s p_{mut} \right) \quad (3.10)$$

In general, this theory predicts that the number of a given schema, N_s , whose average fitness, f_s is far enough above the population average fitness, \bar{f} , will overcome the disruption caused by crossover and mutation and will increase from generation g to generation $g+1$. This is not necessarily a convergence proof, but only a characteristic of the algorithm as it transitions from one generation to the next. Selection pressure is another term that is often used to describe the how well a given chromosome performs when compared to the population as a whole when using a fitness-proportional selection operator.

3.10 Building Block Concept

Closely coupled with John Holland's Schema Theory is the concept of building blocks that was originally introduced and developed by Goldberg [91]. This concept essentially states that fundamental building blocks, or lower order schemata are pieced together by the genetic algorithms in order to move toward the optimal solution. This concept is central to much of the theoretical work relating to population sizing that will be discussed in Section 3.13.2. While this idea is very straightforward and logical, Goldberg also later points out that there are deceptive problems where these building blocks can actually lead the algorithms in the wrong direction. It is also worthwhile noting that the defining length of the critical building blocks can also play a key role in how well a GA will be able to use implicit parallelism to piece together the correct set of building blocks that result in an optimal solution.

3.11 Deceptive Problems

A fundamental question regarding GAs and finding a minimum/maximum point is simply put: what makes a problem difficult for a GA to optimize? Referring back to the Schema Theory and building block concept, a genetic search tends toward low order schemata that seem to produce the best fitness values. It uses these low order schemata as building blocks and works toward higher order schemata that hopefully contain the optimal solution [91]. A problem that often finds itself in these types of discussions is the “needle in a haystack” problem in which the entire search space has a uniform fitness value except for one single point that has the best fitness and is the global optimum [92]. It can be imagined that all of the low order schema would have the same fitness so there would be no information or building blocks to help the GA trend toward the global optimum. The only way for this global optimum to be found would be for the algorithm to land on the exact solution by

random chance. This is of course an issue that is not unique to GAs given that this type of problem is notoriously difficult for many solvers including gradient-based ones [92], [93]. The afore mentioned problem is not necessarily a deceptive problem, but is simply a difficult problem to solve. A deceptive problem is difficult for slightly different reasons in that there are low order schema that have better average fitnesses than the set of low order schema that contains the true optimal solution. In other words, the GA tries to use the problem fitness data to steer the search, but is deceived by the building blocks and ends up trending toward a suboptimal point. In some ways, gradient based methods face similar challenges on multimodal problems where the search may be steered to locally optimal points that are not globally optimal. Goldberg and others have studied this phenomenon in detail and uses Walsh transforms to create deceptive functions and analyze GA performance on these types of problems [94]–[96]. Given the fitness-based selection methods described above, the population in successive generations will tend toward the low order schema that have the best average fitness. This deceptiveness can best be illustrated with a simple definition that Whitley uses in his paper on the principles of deception illustrated in Equation (3.11) [97]. The case of a three bit encoded problem where 1,1,1 is the optimal solution and the string with the maximum Hamming distance from the optimum 0,0,0 is a deceptive attractor, meaning it is a local optimum that creates this deceptiveness. Equation 3.11 illustrates the relationships in schemata fitness that lead to a fully deceptive problem. The schema fitness is the average fitness of all of the strings that lie within the schema (or hyperplane) of interest e.g. $f(0, 0, *) = \frac{f(0,0,0)+f(0,0,1)}{2}$.

$$\begin{aligned} f(0, *, *) &> f(1, *, *) & f(0, 0, *) &> f(1, 1, *), f(0, 1, *), f(1, 0, *) \\ f(*, 0, *) &> f(*, 1, *) & f(0, *, 0) &> f(1, *, 1), f(0, *, 1), f(1, *, 0) \\ f(*, *, 0) &> f(*, *, 1) & f(*, 0, 0) &> f(*, 1, 1), f(*, 0, 1), f(*, 1, 0) \end{aligned} \quad (3.11)$$

It can be seen that all the instances of the lower order schema seem to have higher average fitness values than the low order schema that actually contain the globally optimal solution. This is a fully deceptive problem in that this is the case with all of the lower order schemata. As one can imagine, there are varying degrees of deceptiveness. Unfortunately, a practical metric that enables practitioners to determine deceptiveness and problem difficulty before attempting to solve them for general classes of problems has not yet been found [73]. Whitley tests a few different GA setups against some known deceptive problems to see what works

the best. As it turns out, the GAs with larger populations and more specifically, a parallel GA with distributed populations ends up with the best results in finding the global optimum on deceptive problems [97]. Forrest and Mitchell further investigate this phenomena using Walsh transforms to attempt to pinpoint building block characteristics that may contribute to this deception. They illustrate that simple hill-climbing algorithms and even random search can outperform GAs on some of these deceptive problems [96].

As it turns out, there are measures that can be taken to handle highly deceptive problems. Goldberg, Deb, and Horn investigate a specific type of deceptive trap functions that are extremely multi-modal with over five million local optima and 32 globally optimal solutions to further understand how GAs have difficulty with these types of problems and what can be done to avoid these difficulties [98]. They find that large populations can greatly increase the chances of finding at least one of the globally optimal points. In addition, they investigate the use of niching, a process that penalizes individuals as a function of proximity to other individuals in the population in order to force diversity in the populations. This also helps force the GA to span larger portions of the search space. Niching, in combination with scaling the cost function in a way that forces the globally optimal points become significantly larger (for maximization) than the locally optimal points proved to be the most effective method. Niching alone was not sufficient due to the proximity of global optima relative to each other and relative to local optima that were close in fitness to the global optima. However; the combination of niching, cost function scaling, and larger populations led to GA runs that found all 32 globally optimal points. While these authors had the luxury of knowing that they were dealing with this specific type of multi-modal deceptive function *a priori*, they did illustrate the effectiveness of large populations and population diversity in solving difficult problems. Even with this knowledge, Goldberg, Deb, and Horn had to experiment with different population sizes, different niching distance penalties, and different objective function scaling exponents in order to finally achieve the desired results. Unfortunately, one of the biggest challenges in dealing with difficult problems is first knowing what type of difficulty needs to be dealt with.

Deception is only one factor that can make problems difficult to solve for GAs. In fact, Goldberg identifies five categories of problem characteristics that create difficulty for standard GAs [99].

Isolation

The global optimum is isolated such as in the needle-in-the haystack problem where the global point is an isolated occurrence with a single point that is different from an otherwise uniform fitness landscape.

Deception

Described in the beginning of this section.

Noisiness

When the members of a given schema have a large variance in terms of fitness values. This makes it difficult for the GA to discern which schema has the best average fitness from a limited sample size.

Crosstalk

Goldberg summarizes this as an issue that arises when the higher-level objective function is composed of a combination of sub-level functions. Crosstalk is when there is a cross-coupling affect between sub-functions that does not directly link itself to changes in the encoding structure making it difficult for the GA to optimize the top-level objective function.

Massive Multimodality

The presence of multiple, locally optimal points can cause the GA to prematurely converge to a solution that is not the global optimum.

3.12 Dynamic Modeling and Convergence of Genetic Algorithms

While the Schema Theory is useful in understanding GA population trajectories and behaviors, it can't be used to prove convergence. More recent GA theory has transitioned toward the use of Markov Chains and statistical mechanics to attempt to model GA dynamics and prove convergence. These ideas are investigated in the following sections. In general a Markov Chain is a probabilistic model that can predict the chance of transitioning from one state to another in a sequence where the probabilities are only dependent on the current state and the next state, but are independent of any previous states and transitions. While these methods of using Markov Chains to predict GA dynamics offer the potential to predict simple GA trajectories, they are severely limited to only dealing with simplistic problems, specific GA types, and small search domains [73]. However, more recent theoretical research has been progressing in the right direction toward a robust, mathematically sound

convergence proof that is of value to the practitioner. For this reason, it is worth giving the reader a top-level overview of more recent GA dynamic modeling and convergence theory.

3.12.1 Markov Chain Theory

GAs are stochastic search algorithms where the sequence of successive populations can be assumed to be a finite state space set by fixed chromosome lengths. In addition each current population is only conditionally dependent on the preceding population from the previous generation. Given these two characteristics, GAs fit nicely into a category of algorithms that can be modeled using Markov Chains or series of events that only depend on the state directly preceding the event of interest. A number of researchers have leveraged Markov Chain Theory to further investigate GA convergence in a probabilistic sense, determine upper and lower iteration limits, and even find which problems are likely to be difficult for GAs to solve for some more simplistic examples [81], [100]–[112]. The use of Markov Chain Theory to prove convergence of certain types of GAs began in the early 1990s and continues to be developed and applied in research specific to GA convergence. This section takes the reader though a brief chronological summary of the major developments and contributions in this field. It must be noted that this is only a top-level survey and that further reading is required to obtain details and derivations for the following convergence models.

In the early 1990s, Vose and Liepins initially set the stage by deriving the equation used to calculate $p_i(y)$ the probability of creating chromosome y from a population accounting for fitness-proportional selection, single-point crossover, and mutation [113]. Their interest was to investigate punctuated equilibrium points where the GA tends to temporarily stagnate during the course of its evolution. This model for a simple GA became a critical stepping stone that numerous authors use to further predict GA dynamics using Markov Chains. Vose and Liepins first define $m_{i,j}(0)$ as the probability of parents i and j producing a chromosome of all zeros based on the single point crossover rate, p_{cross} , and bit-wise mutation rate, p_{mut} in Equation 3.12. It must be noted that \oplus is the exclusive OR operator and \otimes is the logical

AND operator.

$$m_{i,j}(0) = \frac{(1 - p_{mut})^l}{2} \left\{ \eta^{|i|} \left(1 - p_{cross} + \frac{p_{cross}}{l-1} \sum_{k=1}^{l-1} \eta^{-\Delta_{i,j,k}} \right) + \eta^{|j|} \left(1 - p_{cross} + \frac{p_{cross}}{l-1} \sum_{k=1}^{l-1} \eta^{\Delta_{i,j,k}} \right) \right\} \quad (3.12)$$

where l is the length of each binary chromosome, $\Delta_{i,j,k} = |(2^k - 1) \otimes i| - |(2^k - 1) \otimes j|$, and $eta = \frac{p_{mut}}{1-p_{mut}}$. Vose and Liepins let $M_{i,j}$ be a matrix whose elements represent the i th and j th components of $m_{i,j}(0)$. Further they define a permutation column vector as $\sigma_j \langle x_0, \dots, x_{r-1} \rangle^T = \langle x_{j \oplus 0}, \dots, x_{j \oplus (r-1)} \rangle^T$ with \bar{x} being a chromosome in a given population. Putting these two definitions together, they create an operator that accounts for mutation and crossover as seen in Equation (3.13).

$$\mathcal{M}(\bar{x}) = \langle (\sigma_0 \bar{x})^T M \sigma_0 \bar{x}, \dots, (\sigma_{r-1} \bar{x})^T M \sigma_{r-1} \bar{x} \rangle^T \quad (3.13)$$

where, $r = 2^l$ is the total number of possible unique chromosomes that can be created using binary representation. Given $\mathcal{M}(\bar{x})$, the authors develop a selection operator that is based on fitness values to complete their model for $p_i(y)$. In order to do this, they first define a diagonal matrix F where $F_{i,i} = f(i)$ with $f(i)$ being the fitness value of chromosome i . Next, they define an incidence vector (ϕ_i) that is of length 2^l with an element for each possible chromosome. Each element of ϕ_i represents the number of times that specific chromosome occurs in population P_i . Combining these ideas with $|F\phi_i|$ being the element-wise sum, Equation (3.14) illustrates the probability of producing chromosome y from population P_i .

$$p_i(y) = \mathcal{M} \left(\frac{F\phi_i}{|F\phi_i|} \right)_y \quad (3.14)$$

After this important contribution, Nix and Vose were some of the first to model simple GAs using Markov Chain Theory in a way that doesn't assume infinite populations [109]. In order to create the probability transition matrix $Q_{i,j}$, the authors assume a simple GA that implements bitwise mutation, selection probability that is proportional to the chromosomal fitness, and single point crossover. Referring to Equation (3.16) l is the length of each chromosome, N_{pop} is the number of chromosomes in a population, $r = 2^l$ is the number of

unique chromosomes assuming binary representation, and N_{tot} is the number of possible combinations of chromosomes in a given population (order doesn't matter) as depicted in Equation (3.15).

$$N_{tot} = \binom{N_{pop} + r - 1}{r - 1} \quad (3.15)$$

They let Z be a N_{tot} by r matrix where each element describes the number of specific chromosomes, r , in each unique population, N_{tot} , in other words Z represents the state space. Given this, the state transition matrix $Q_{i,j}$ is an N_{tot} by N_{tot} matrix that describes the probability of transitioning from one population P_i to another population P_j . It must be noted that the size of Q becomes unmanageable very quickly, making it hard to calculate for realistically sized populations and string lengths. As an example for a population of 10 with strings of 5 bits, N_{tot} is 1,121,099,408. Nix and Vose calculate each element of the state transition matrix using Equation (3.16). The probability function $p_i(y)$ describes the probability of producing a given chromosome y from a given population P_i as a function of chromosome fitness, crossover probability, and mutation probability. Nix and Vose leverage the work of Vose and Liepins [113] who originally derive the $p_i(y)$ function that is used to find the probability for a population to produce a given chromosome based on selection, mutation and crossover as depicted in Equation 3.14.

$$Q_{i,j} = N_{pop}! \prod_{y=0}^{r-1} \frac{[p_i(y)]^{Z_{y,j}}}{Z_{y,j}!} \quad (3.16)$$

From this, the probability vector that illustrates the probability of each chromosome being present in the population after g generations can be found using $\pi = P_0 Q^g$, where P_0 is a vector of length N_{tot} with a 1 indicating the presence of a given chromosome in the initial population and a zero otherwise. With this Nix and Vose create an exact GA Markov Chain model by using this $p_i(y)$ function. This model is then used to investigate and prove asymptotic behavior to show that certain steady states can be reached as the number of generations moves toward infinity with sufficiently large populations and that the GA will not be trapped at a suboptimal point given that $Q_{i,j}$ is ergodic [113]. Ergodic in this case essentially means that any state can eventually be reached regardless of the algorithm's initial state. In GAs, this characteristic is achieved by the mutation operator when $p_{mut} > 0$.

From here, Rudolph illustrates that not all canonical GAs will converge to a global optimum but rather only ones that memorize the best solution found or utilize elitism [81]. These convergence properties are again illustrated through the application of homogeneous, finite Markov Chain Theory. Rudolph proves the following theorem regarding the global convergence of simple, elitist GAs. While this idea is not particularly useful to the practitioner, it continues to be the backbone of convergence proofs and ideas associated with these types of algorithms.

Theorem: [81] Let be the population P_t at time t and let $f(\bar{x}_{best}(t))$ be the fitness of the best solution in P_t . If a genetic algorithm has mutation probability $0 < p_{mut} < 1$ arbitrary crossover and selection, but is elitist, then the sequence $D_t = f^* - f(\bar{x}_{best}(t))$ (where f^* is the optimal solution) is a non-negative supermartingale which converges almost surely to zero [73].

In this theorem, a non-negative supermartingale indicates that $E[D_{t+1}|\bar{x}_{best}(t)] \leq D_t$ meaning that the difference between the global optimal solution and the best solution in each successive generation is decreasing, or at worst remains unchanged from one generation to the next [73]. In theory, with a non-zero mutation rate, the algorithm has a non-zero probability of escaping any locally optimal point where the algorithm may temporarily stagnate for a number of generations.

De Jong, Spears, and Gordon point out that most discussions of convergence and problem difficulty specific to GAs do not take into account GA type, encoding, and several other factors that could significantly change the results and problem difficulty [114]. They note that for GAs with non-zero mutation rates, every point in the search space has a positive probability of being searched as the number of generations approaches infinity, no different than a purely random search technique. Given this, infinite time global convergence proofs are not necessarily useful, however, there are some questions regarding typical measures of effectiveness that would be more useful in determining how ‘easy’ or ‘difficult’ a problem is for a given GA configuration. Typical measures of GA progress consist of the population mean fitness value vs. generation number, the best individual found to date vs. generation number, or the best individual found in the current population vs. the generation number. They add to Nix and Vose’s work by focusing on the transient behavior of the Markov Chains in an attempt to answer questions relating to the probability of an optimal solution being

in a given generation, or the mean time it will take to find an optimal solution. They use illustrative examples to show how these calculations work for deceptive and non-deceptive problems. It turns out that these are some of the better metrics for determining whether or not a problem is “GA-hard” or not. At the end of their work, they look at mutation rates, crossover rates, and objective function scaling to gain some insight into optimal settings for these GA attributes. Using Markov Chains, they are able to compare differences between deceptive problems and non-deceptive problems regarding probability of finding the optimal solution vs. the number of generations and wait time before the optimal solution is found. Of course, this analysis had to be done on relatively small problems so that the state transition matrices Q could be determined for each case.

After this, Aytug and Koehler carry this idea a little further in establishing an upper and lower bound on the number of generations required to find the optimal solution for a given confidence level, δ , with a given mutation rate, p_{mut} [102]. The upper bound of the expected number of generations, g , required to find the optimal solution for a given confidence level can be seen in Equation (3.17). It must be noted that this doesn’t take into account any information regarding the optimization problem characteristics. The only parameters that the user can tune in order to impact this bound are the GA population size, N_{pop} , the mutation rate, p_{mut} , the chromosome length, l , and the desired confidence level, δ .

$$E[g(\delta)] \leq \left\lceil \frac{\ln(1 - \delta)}{\ln(1 - \min\{(1 - p_{mut})^{N_{pop}l}, p_{mut}^{N_{pop}l}\})} \right\rceil \quad (3.17)$$

From this point, Greenhalgh and Marshall point out that ensuring every potential population must be seen in order to find the optimal solution is overkill when really all that is needed is that every potential chromosome is seen [106]. Given this, these authors are able to prove that Equation (3.18) provides a much tighter and much more realistic upper bound on the number of iterations required to see the optimal solution with a given confidence level. It must be noted that the above methods for probabilistic convergence properties assume GAs that use some type of elitist strategy, or at least keep a memory of the best solution seen across all generations. Without this characteristic, the optimal solution could be seen and then forgotten in later generations. It can also be seen that Greenhalgh and Marshall don’t limit the application to binary encodings where the cardinality (K) is equal to 2, but instead

have created a model that can account for higher cardinality encodings. Given this, it can still be seen that the only tunable parameters are population size, N_{pop} , chromosome length, l , mutation rate, p_{mut} , confidence level, δ , and now encoding cardinality, K . Crossover probability and problem specific characteristics still do not factor into this upper bound on required generations. This is due to the originating convergence proof being entirely based on random search aspects of the GA that are produced by the mutation operator.

$$E[\bar{g}(\delta)] \leq \left\lceil \frac{\ln(1 - \delta)}{N_{pop} \ln \left[1 - \min \left\{ (1 - p_{mut})^{l-1} \left(\frac{p_{mut}}{K-1} \right), \left(\frac{p_{mut}}{K-1} \right)^l \right\} \right]} \right\rceil \quad (3.18)$$

As can be seen from the previous overview of Markov Chain methods, there has been a significant amount of progress in modeling GAs and developing models to estimate their dynamics. However, it can also be seen that these methods are still very limited in their applicability to real-world problems and search domains due to the curse of dimensionality. In addition, these methods do not account for problem-specific characteristics that could ultimately influence the dynamics and convergence properties of these algorithms. Several authors have investigated the idea of grouping states to somewhat mitigate the curse of dimensionality with limited success [111]. Along the same lines, researchers are also developing statistical mechanics approaches that also attempt to address the prohibitively large amounts of data needed to fully model GAs as Markov processes.

3.12.2 Statistical Mechanics

One of the major shortfalls in using Markov Chain approaches for analyzing GAs is that the curse of dimensionality quickly manifests itself, given that the probability matrices that describe the chance of transitioning to one population from another have to be exhaustive in terms of describing all of the possibilities [73]. This makes these methods impractical for anything but a few oversimplified problems and GA algorithms. With this, a lot effort has been put forth to develop approximations methods to model GA dynamics. Reeves and Rowe use the physics-based analogy of predicting how gas molecules in a balloon will behave [73]. While it is impossible to track every molecule, approximation methods have enabled physicists to successfully model these dynamics and properties by reducing the

number of parameters used to describe these characteristics. Several researchers have been working on applying these same types of statistical mechanics techniques to GAs [115], [116].

These statistical mechanics methods essentially use a Taylor Series expansion to approximate the population distribution function. The simplification occurs in only utilizing the first several terms, which prove to be fairly accurate when the population fitness distributions are close to being Gaussian. With this idea, the probability models that result from selection and simple bitwise mutation are fairly easy to generate regardless of the fitness function [73]. However, crossover is fairly challenging and requires that there is close linkage between the chromosome values “genotype” and the fitness values “phenotype.” As an example, a function such as the appropriately named onemax problem where the fitness value is simply the sum of all of the ones in the chromosome, is an ideal function for this method. Very accurate dynamic predictions for a simple GA solving the onemax problem have been obtained when small sample sizes have been accounted for. Unfortunately, because of the requirement to have a direct linkage between “genotype” and “phenotype,” these statistical mechanics techniques have only been successfully applied to a select few simplified test functions [73]. This limitation makes these statistical analysis approaches somewhat lacking in terms usefulness to the GA practitioner, however marks significant steps toward a more practical dynamic model.

3.13 Genetic Algorithm Parameter Settings

A question that quickly arises when looking at GAs is what are the appropriate parameter settings to achieve efficient GA performance. There are a large number of tunable parameters such as: population size, number of bits per variable, probability of mutation, crossover probability, pool size (tournament selection), maximum number of generations, etc. that can quickly become overwhelming for a user to appropriately tune. One of the more practical studies on GA parameter settings was an analysis of variance (ANOVA) study conducted by Rojas et al. that evaluated the amount of impact that a number of user-defined parameters had on the solution quality obtained for several test functions [117]. More specifically, the test functions included six problems from literature covering several types of function classes including: combinatorial, continuous, multimodal, and functions with known deceptiveness. This ANOVA experiment considered GA factors such as: mutation

rate, crossover rate, crossover type, selection type, number of generations, and population size. The ANOVA study shows that population size and selection type have the largest impact on the best fitness values found, but that all factors have a statistically significant impact on GA performance. Rojas et al. also conclude that the optimal tuning of these parameters is problem dependent and is still an open area of research.

3.13.1 Crossover and Mutation Rates

De Jong was the first to study simple GA parameter values in his dissertation work and uses several metrics and ideas to develop the first documented set of appropriate parameter values for a simple GA [68]. He investigates the effects of population size, mutation rate, and crossover rate on several metrics to include on-line performance, off-line performance, and allele loss. On-line performance evaluates how the fitness progresses at each generation while off-line performance only looks at the generations where the best fitness value improves. Allele loss, investigates the premature loss of alleles, or specific schema, as a factor that can contribute to premature convergence. In several of De Jong's evaluations, $p_{mut} = 0.001$ and $p_{cross} = 0.6$ perform the best, however, he notes that no one combination of operator values (within a reasonable range) significantly improved or degraded the performance of the GAs used in his research. As a rule of thumb, De Jong indicates that the mutation rate should be approximately $p_{mut} = \frac{1}{\text{string length}}$ [68]. Grefenstette takes this analysis of crossover and mutation rates one step further by utilizing genetic algorithms to select the best GAs in terms of combinations of parameters [118]. He sets up an experiment that essentially runs a population of 1,000 variations of genetic algorithms in terms of six parameters including: population size, mutation rate, crossover rate, generation gap, scaling window, and selection strategy. Where the generation gap describes the percentage of the population that evolves vs. a percentage that is preserved from one generation to the next. The scaling window consists of how much the fitness values are scaled to adjust the relative difference between chromosomal fitness values in a population. Lastly, the selection strategy consisted of selection either with or without the use of elitism. A top level GA then determines the best GAs for a given problem set of five optimization problems for both on-line and off-line performance. For on-line performance, Grefenstette's top level GA found that a GA with $N_{pop} = 30$, $p_{mut} = 0.01$, $p_{cross} = 0.95$, $\text{generationgap} = 1.0$, $\text{scaling} = 1.0$ and elitism was found to be the best performing. For the off-line test, the GA with $pop = 80$, $p_{mut} = 0.01$, $p_{cross} = 0.45$, $\text{generationgap} = 0.9$, $\text{scaling} = 1.0$ and no elitism

was found to be optimal. It must be kept in mind that these empirical results pertain to the specific problem set, bit resolution, and other attributes that Grefenstette utilized for this study. There are some general conclusions, however, that add to the body of knowledge in terms of appropriate settings for GA parameters. In concurrence with De Jong, Grefenstette concludes that mutation rates that are higher than 0.05 seem to cause too much disruption and degrade GA performance, while GAs without mutation also perform poorly when compared to ones that do. In addition Grefenstette finds that a large generation gap e.g. large portions of the population undergoing evolutionary operations seems to produce better results (e.g. higher crossover rates). Lastly, elitism improves performance and convergence speeds in most cases. Across literature, typical ranges for fixed crossover parameter settings are $p_{mut} = [0.001 - 0.05]$ and $p_{cross} = [0.5 - 1.0]$ [86]. For the purposes of the practitioner, these ranges should be utilized as starting points when attempting to optimize a new problem.

A number of studies have been conducted looking at different self-adaptive schemes used to adjust the parameters in genetic algorithms. Srinivas et al. explore the transitions between exploitation and exploration by adapting both mutation rate and crossover rate as a function of how the fitness values are progressing over time [86]. Equation (3.19) illustrates that their method is setup to work on maximization problems in two ways. First, it tries to preserve the best chromosomes. Given that f' is the better fitness of the two parents and k_i are user-defined learning parameters, then the crossover/mutation rates will be smaller when $f(\bar{x})$ and f' are high for a maximization problem. This is a clever way of not disrupting chromosomes that are performing well compared to the best fitness found so far, f_{max} . Next, this method increases the rates when the GA has converged. They anticipate that the population average fitness, \bar{f} , will approach the best fitness found in an elitist GA which is why the difference between the two values is in the denominator.

$$\begin{aligned}
 p_{mut} &= k_1 \frac{f_{max} - f(\bar{x})}{f_{max} - \bar{f}} \\
 &\text{if } \bar{f} \geq f(\bar{x}), \quad p_{mut} = k_3 \\
 p_{cross} &= k_2 \frac{f_{max} - f'}{f_{max} - \bar{f}} \\
 &\text{if } \bar{f} \geq f', \quad p_{cross} = k_4 \\
 (k_1, k_2, k_3, k_4) &\leq 1.0
 \end{aligned} \tag{3.19}$$

After appropriately choosing k_i values through trial and error, the authors illustrate that this adaptive method performs favorably when compared to fixed rates in a standard GA configuration.

Thierens uses the onemax problem to compare different adaptive, dynamic, and fixed mutation techniques [119]. He illustrates that the adaptive mutation rate techniques can achieve superior results when compared to the fixed mutation rates on two test problems. In addition, their decreasing gain adaptive method produces similar results to the simulated annealing style, exponentially decreasing mutation rate technique. These findings are later applied in Section 4.5 of this dissertation to develop new RCGA crossover techniques.

3.13.2 Population Sizing

The question of optimal population sizing for GAs is a fundamental one that is very closely linked with GA performance and convergence properties. Unfortunately, this question of optimal population sizing has proven to be a fairly difficult question to answer. Like many aspects of genetic algorithm attributes, the optimal population size turns out to be somewhat problem dependent. In addition to this, there is no agreed upon standard for sizing the population for any given problem and GA type. Instead, there are a number of empirically generated guidelines regarding population size and a handful of theoretically based population sizing studies and rules have been developed throughout literature. In application, some practitioners point to simple rules such as the population should be approximately 10 times larger than the number of decision variables or that the population should simply be around 50 [120].

Goldberg spent a good deal of effort employing John Holland's Schema Theorem and Goldberg's related building block hypothesis toward the derivation and understanding of appropriately sized populations for GAs [121]–[123]. Goldberg et al. are some of the first to derive a formula that describes the optimal population size for a given problem [123]. Their logic and derivation is heavily based on Holland's schema theorem and the closely related building block concept. They leverage the idea that the GA is solving a large number of multi-arm bandit problems in a way that determines the best building blocks over time that eventually lead the algorithm to an optimal solution. Although counterexamples to this idea have been illustrated, the idea hasn't been entirely dismissed. Goldberg et

al. propose that noise surrounding these building block competitions is one of the major factors leading to convergence. With this assumption in mind, they are able to statistically prove that noise decreases as population size increases. Using this fundamental idea, they illustrate that population size should grow in an exponential way as the string length or problem size is increased. Goldberg et al. verify this idea using five test functions with varying string lengths and population sizes. They indicate that this is likely a conservative population sizing relationship, as is also pointed out in several other papers that reference this work [73], [120].

Herick et al. model a simple selection operator in a GA as a one-dimensional random walk. This enables them to rely on well-founded stochastic theory to derive a model for GA convergence as a function of population size [124]. More specifically, Herick et al. model competitions between building blocks as being analogous to the gambler's ruin problem. Using this approach, they are able to derive a fairly simple model to determine appropriate population sizes. Using these assumptions, Herick et al. indicate that the population should grow with the square root of the problem size. Equation (3.20) illustrates the end result where the population size is a function of the building block length, k_{bb} , the chromosome length, l , the number of correctly chosen building blocks, Q_{bb} , the standard deviation of fitness values of chromosomes containing the building block, σ_{bb} , and the difference in mean fitness between the two competing building blocks d_{bb} . It is quickly seen that this type of information is available only on simple problems where building blocks are easily identified, such as the onemax problem that several of these papers use to illustrate their models. While this idea is interesting in theory and potentially insightful in terms of the general growth rate required for population sizes as the chromosome lengths are increased, it isn't very practical in application due to the required information that is not likely available when solving real-world engineering problems. In addition, the above methods are all anchored on the schema theory and building block concepts which have been brought into question through several papers and counterexamples that don't seem to follow what these theories suggest [90], [125].

$$N_{pop} = -2^{k_{bb}-1} \ln \left(1 - \frac{Q_{bb}}{l} \right) \frac{\sigma_{bb} \sqrt{\pi(l-1)}}{d_{bb}} \quad (3.20)$$

A more sophisticated approach may be to refer back to Equation 3.18 and recall that the upper

bound on the required number of generations to reach a confidence level δ is proportional to population size. More specifically, a larger population results in a lower number of required generations to reach the optimal solution. However, it must be noted that larger populations also require a larger amount of computation time to evaluate the associated increase in function evaluations and genetic operator runs. In addition, it has been shown that larger numbers of decision variables require more generations for convergence [73]. However, most authors also indicate that larger bit strings or larger numbers of decision variables require larger populations based on the idea of sampling noise and schema representation. This further illustrates that the optimal population size is not a straightforward calculation, especially when considering problem-specific characteristics.

Given this difficulty to perfectly optimize population size for any given problem based on theory alone, many of the authors utilize these theoretical approaches as starting points to then empirically study population size. As an example of a theoretical based guideline, Reeves and Rowe define a lower bound on population size so that the population is large enough to enable any possible chromosome being constructed from crossover alone (no mutation) [73]. Equation 3.21 illustrates that this number is essentially an exponential function that creates a very optimistic outlook on population requirements.

$$N_{pop} = \left\lceil 1 + \frac{\log\left(\frac{-l}{\ln\left((1 - (\frac{1}{2})^{N-1})^l\right)}\right)}{\log(2)} \right\rceil \quad (3.21)$$

Tsoy runs a series of test problems to determine some characteristics between population size and the limit on number of generations. [126] In general, his findings indicated that better results are achieved with larger populations, even with fairly small generation limits when compared to smaller populations with large generation limits. Alander uses empirical data from five test problems to determine that the optimal population size seems to be equal to the number of bits per chromosome string [120].

It can be seen that optimally sizing a population for a given type of GA on a given problem requires a combination of theoretical guidelines coupled with problem-specific tuning. In addition, the use of parallel processing can also influence the optimal population size where

the computational penalty of increasing population size becomes somewhat mitigated. These are important aspects to keep in mind as new GA operators are applied to new problems using parallel architectures in the following chapter.

3.14 Binary Genetic Algorithm Improvements

A number of improvements and adaptations to the original binary GAs introduced by Holland and De Jong have been developed and applied to overcome a number of issues and shortfalls observed by many GA practitioners. At the top level, these adaptations have been utilized to overcome challenges associated with isolation, multimodality, deception, and noisiness [99]. Several of these are introduced in this chapter simply so the reader has an idea of some of the tools available to address different types of optimization problems. They are also discussed to help inform new ideas and innovations that will later be developed and applied to optimal control problems in the following chapter.

3.14.1 Messy Genetic Algorithms

It is interesting to note that GAs have primarily been consistent over the years in following the idea that chromosome lengths and population sizes should be fixed. However, Goldberg et al. illustrate the potential benefits of more closely following biological processes and violating these standard fixed designs as they develop the messy GA [127]. Messy GAs as introduced by Goldberg et al. are entirely built around the building block concept and used to overcome the difficulties associated with deceptive problems. These algorithms operate in two phases where the initial population contains all of the possible building blocks of a certain order. This requires a fairly large population, and knowledge of the defining length of building blocks. The initial phase of the algorithm simply refines the set of building blocks to identify the best ones. During the second phase, the population begins shrinking as the building blocks are pieced together to form variable length chromosomes. This algorithm is initially put to the test on a deceptive type problem where sets of three bits are able to be individually evaluated and summed together to determine the overall fitness of a given chromosome. On this specific type of deceptive problem that has been designed to support the building block, or separable fitness function idea, the messy GA significantly outperforms the standard GA. In a later paper, Goldberg et al. further improve upon this idea to decrease the required size of initial population and slightly change the way the algorithm

works [99]. This fast messy genetic algorithm is able to reduce the initial population size considerably by using longer chromosomes that are close in length to the problem string length. This enables all of the building blocks to still potentially be considered without a very large population of short chromosomes. These longer chromosomes are then refined down to the essential building blocks with the deletion of lower performing genes and selection of higher performing ones during the first phase of the algorithm. After this initial population has been refined down to the best building blocks, the algorithm begins the second phase and combines these building blocks in different ways to arrive at a solution. This type of messy GA is again tested on deceptive problems using a combination of fifth order deceptive functions that are pieced together to form larger trap functions. The messy GAs again outperform the standard GAs and the fast version finds the optimal solution more rapidly than the original messy GA described above. These messy genetic algorithms appear to be good at solving certain types of deceptive problems that other GAs have difficulty solving. However, they have not been proven to perform better across the board, and for this reason have not been a very commonly used GA when looking at engineering application problems.

3.14.2 Dynamic Resolution

One of the most notable challenges that GAs face is premature convergence to a local minimum in multimodal problems. Andre et. al. investigate this phenomenon by starting with the most basic binary genetic algorithm [128]. They then improve this algorithm by using reflected binary gray coding and two-point crossover. However, they notice that while speed is improved with this algorithm, the issue of premature convergence is still experienced on a number of the multi-modal test problems. Andre et. al. introduce two interesting ideas to help remedy this issue of converging to local optima. The first technique is to adjust the search range or upper and lower bounds on the variables based on the top performing chromosomes. This is analogous to zooming in so that the search effort can be applied to a much higher resolution, but smaller search area. After a set number of search iterations have been conducted in the zoomed-in search area, the algorithm then zooms back out to the initial problem bounds. After each time this is done, the population is initialized and the algorithm begins again. The second improvement that they added was a scale factor that impacts the relative fitness of the best chromosomes compared to the worst ones. The scale factor is adjusted in a way that allows the poor solutions to have a better chance of being

selected earlier in the algorithm than they do at the end of the GA run. This is done to help ensure a non-homogeneous population is maintained during the early stages of the GA to improve exploration properties. Andre et. al. then test this algorithm on 20 benchmark test problems and observe fairly significant improvements on some of the multimodal problems that the simple GA was having difficulty with. However, they did note that the algorithm still had trouble finding optimal solutions for problems that had high numbers of decision variables. In addition, Andre et. al. point out that the added improvements do come with the cost of additional computation time when compared to the simple GA in terms of the amount of time required to reach the same number of generations.

3.14.3 Niche Formation

Another adaptation that can be used to help find globally optimal solutions, or at least some of the best locally optimal solutions on multimodal problems is a technique that parallels biological concepts by forming niche populations that group around different local optima. This idea is initially introduced by De Jong [68], and is further developed and employed by several other authors [98], [129]–[132]. De Jong initially introduces the idea of crowding where overlapping populations only allow a certain number of their members to reproduce during each generation. In addition, chromosomes are allowed to die off or be replaced by a new chromosome based on how similar they are to the new chromosome. The old chromosome that is the most similar to the new chromosome is then replaced by the new one, forming niches of similar populations that evolve and group around different local optima. Goldberg et al. develop a slightly different method of niche formation using a technique that they refer to as sharing [129]. The sharing method also uses subpopulations and determines the relative similarity of chromosomes in terms of decoded phenotype, or in terms of encoded genotype using various techniques. Given this metric, the algorithm then penalizes solutions if there are a large number of chromosomes that are within a certain neighborhood of that given solution. This is another way of forcing population diversity and forcing the algorithm to spread out across the search space. Deb et al. investigate the performance of two different sharing techniques and De Jong’s crowding technique to see which one performs the best on several multimodal functions [130]. Their empirical analysis indicated that the sharing techniques do the best job of locating all of the optimal points over the three multimodal problems used for the analysis. They also discover that the Hamming distance or genotype similarity measure doesn’t perform as well when there

are local optima that are of unequal value, but performs better than the phenotype distance measure on problems where all of the local optima are of equal fitness.

3.14.4 Scaling

Scaling consists of changing the scale of the original fitness function in order to assist the algorithms in distinguishing between local optima that may otherwise be very close in fitness value. This technique is successfully demonstrated by Goldberg et al. in their paper investigating techniques to help GAs solve massively multimodal, deceptive functions [98]. They demonstrate that scaling can make the difference between finding the global optimum and a locally optimum point in the deceptive problem of interest. Grefenstette investigates a slightly different scaling mechanism [118]. He represents the fitness of a given maximization problem solution vector \bar{x} by looking at the difference between the minimum objective function value in the population f_{min} and the current evaluated fitness function $f(\bar{x}) - f_{min}$. It is noted that the selection pressure significantly decreases in a population where all of the fitness values are close by comparison using this technique. Given this, he defines a scaling window term and empirically studies its effects on GA performance. His technique then subtracts an appropriate value from all of the fitness values so that the relative differences between chromosome fitness is large enough to achieve a sufficient selection pressure for a fitness proportional selection scheme such as roulette selection. Empirical results indicate that this scaling process slightly increases GA performance when compared to standard, unscaled versions. Scaling, of course is something that comes up over and over again when dealing with numerical solvers. It is not surprising that this topic is also of great interest when evaluating constraint handling techniques such as the various penalty methods described in Appendix B.

3.15 Real Coded Genetic Algorithms

The previous sections all refer to GAs in their initial binary form as introduced by Holland back in 1975. While a number of improvements, adaptations, and tweaks have been discussed, these all link back to the standard binary coded GA. A RCGA is different in that it uses real numbers to represent each decision variable for a given problem, rather than encoding them using various cardinality alphabets. Holland's Schema Theory, Goldberg's Building Block Theory, and Implicit Parallelism ideas lead to the rule of minimal alpha-

bets, where the ideas behind schema processing are used to prove that the largest number of schema are processed (maximum implicit parallelism) when minimum alphabet encodings (binary encodings) are used [39], [133]. This suggests that binary encodings should achieve superior performance when compared to higher cardinality encodings, especially in comparison to RCGAs which have the highest possible cardinality. However, GAs were originally introduced and applied to combinatorial type problems where binary encodings made the most natural and intuitive sense. In addition, binary encodings also had a natural fit when coupling these algorithms and ideas to a Darwinian, biology-based design model, which is how they were originally contrived. Given this natural application, it makes sense that RCGAs quickly became popular for similar intuitive reasons when applying GAs to optimization problems pertaining to continuous search spaces. Furthermore, there have been a number of empirical studies that seem to illustrate results that are counter to the theory of minimal alphabets in that the RCGAs consistently outperform binary GAs in problems with larger numbers of continuous decision variables [134]–[136].

Even though GAs were first introduced in 1975, it is interesting to note that the first documented GAs that utilized real-coded variables for optimizing continuous functions did not begin to appear until 1989 with Davis adapting GA operator probabilities in real-time and with Lucasius et al. solving chemometrics problems [69], [70]. Since this initial introduction, RCGAs have been applied to a number of problems and have grown in popularity for several reasons. First of all, many users find it more intuitive to have one gene representing one decision variable instead of having multiple genes representing a single decision variable in binary GAs [137]. Another reason is that real-coded variables have the ability to cover large search domains with high precision, where binary encoded variables lose precision (or resolution) as the search domain is increased due to quantization error. The only way around this, for binary GAs, is to use larger and larger binary chromosomes to represent the decision variables, which can have a negative impact on performance [134]. Next, the higher cardinality alphabets are less vulnerable to the afore mentioned issues associated with Hamming Cliffs. Given that RCGAs have the highest possible cardinality, they have fewer low-order schemata and therefore are less likely to experience deception [137]. Real encoding also opens up the trade space in terms of the various crossover and mutation operators that can be used, including methods that have been used on ESs. [135] Lastly, without the extra computational steps of encoding and decoding, the RCGAs tend

to run faster than their binary counterparts [134], this quickly becomes apparent in the following chapter.

3.15.1 Real Coded GA Crossover Operators

There are a number of various crossover operators that have been developed for both RCGAs and ESs. As one can imagine, the standard crossover operators for binary coded GAs described above have severe limitations when directly applied to RCGAs. The primary one being that the decision space is severely limited by the fact that all parameter values would have to be included in the initial population. In other words, each decision variable value would remain unchanged throughout the algorithm. The only change from generation to generation as a result of crossover would be the combination of how these decision variable values appear in a single chromosome. Given this, a number of new crossover operators have been introduced. While this is not an exhaustive list, the more popular ones are discussed in this section.

Simple, single point crossover for RCGAs is first introduced by Wright and is coupled with a fairly high mutation rate to overcome this previously stated drawback of only working in the decision space described by the initial population [135]. Given that there are two parent chromosomes $P^{(1)} = [c_1^{(1)}, c_2^{(1)}, \dots, c_n^{(1)}]$ and $P^{(2)} = [c_1^{(2)}, c_2^{(2)}, \dots, c_n^{(2)}]$ simple one point crossover creates two offspring by swapping consecutive genes between the parents about a uniformly random point in the chromosome. For example if the crossover point is three and the problem dimension, n , is six then the resulting offspring would be $O^{(1)} = [c_1^{(1)}, c_2^{(1)}, c_3^{(1)}, c_4^{(2)}, c_5^{(2)}, c_6^{(2)}]$ and $O^{(2)} = [c_1^{(2)}, c_2^{(2)}, c_3^{(2)}, c_4^{(1)}, c_5^{(1)}, c_6^{(1)}]$. In a similar light, multi-point crossover and uniform crossover can also be applied to RCGAs. These methods all require two parents and produce two offspring as a result.

Flat crossover, first introduced in 1991 as a crossover method that produces offspring decision variables, O_i by utilizing a uniformly random sample between upper and lower bounds defined by the maximum and minimum decision variable values of the parent chromosomes as illustrated in Equation (3.22) [138]. This crossover operator, along with several others can produce as many offspring as desired by taking additional samples from this uniform distribution and are not limited to only two. Given this, there are versions of these types of algorithms that then require an offspring selection operation. Any of

the above described selection methods can be used to pick the two best offspring from the group. After this is done, this operator can be used in a standard RCGA.

$$\begin{aligned}
O_i^{(1)} &\in [c_{min,i}, c_{max,i}] \quad (\text{uniform random}) \\
O_i^{(2)} &\in [c_{min,i}, c_{max,i}] \quad (\text{uniform random}) \\
c_{max,i} &= \max(c_i^{(1)}, c_i^{(2)}) \\
c_{min,i} &= \min(c_i^{(1)}, c_i^{(2)}) \\
\forall i &= [1, \dots, n]
\end{aligned} \tag{3.22}$$

BLX- α is a crossover technique that allows for the offspring to be uniformly selected from a set that extends further than the set created by the parents $[P^{(1)}, P^{(2)}]$ [139]. This technique is further described in Equation 3.23, where I_i is the difference between the largest and smallest of each decision variable and α is a uniform random number between zero and one. The offspring are then randomly selected from a uniform distribution defined by the upper, $c_{max,i}$, and lower, $c_{min,i}$, parameters coupled with the random variable, α . This crossover operator can be further extended to use multiple parents and create multiple offspring.

$$\begin{aligned}
O_i^{(1)} &\in [c_{min,i} - I_i\alpha, c_{max,i} + I_i\alpha] \quad (\text{uniform random}) \\
O_i^{(2)} &\in [c_{min,i} - I_i\alpha, c_{max,i} + I_i\alpha] \quad (\text{uniform random}) \\
c_{max,i} &= \max(c_i^{(1)}, c_i^{(2)}) \\
c_{min,i} &= \min(c_i^{(1)}, c_i^{(2)}) \\
I_i &= c_{max,i} - c_{min,i} \\
\alpha &\in [0, 1] \quad (\text{uniform random}) \\
\forall i &= [1, \dots, n]
\end{aligned} \tag{3.23}$$

BLX- α - β is a slight modification to BLX- α in that the upper end of the set is extended past the original set by a different amount than the lower end of the set [140]. This is further

illustrated in Equation 3.24.

$$\begin{aligned}
O_i^{(1)} &\in [c_{min,i} - I_i\alpha, c_{max,i} + I_i\beta] \quad (\text{uniform random}) \\
O_i^{(2)} &\in [c_{min,i} - I_i\alpha, c_{max,i} + I_i\beta] \quad (\text{uniform random}) \\
c_{max,i} &= \max(c_i^{(1)}, c_i^{(2)}) \\
c_{min,i} &= \min(c_i^{(1)}, c_i^{(2)}) \\
I_i &= c_{max,i} - c_{min,i} \\
\alpha &\in [0, 1] \quad (\text{uniform random}) \\
\beta &\in [0, 1] \quad (\text{uniform random}) \\
\forall i &= [1, \dots, n]
\end{aligned} \tag{3.24}$$

Linear crossover uses two parents to create three offspring that lie between the parents in an evenly spaced, linear fashion as described in Equation 3.25 [135].

$$\begin{aligned}
O_i^{(1)} &= [\frac{1}{2}c_i^{(1)} + \frac{1}{2}c_i^{(2)}] \\
O_i^{(2)} &= [\frac{3}{2}c_i^{(2)} - \frac{1}{2}c_i^{(1)}] \\
O_i^{(3)} &= [-\frac{1}{2}c_i^{(2)} + \frac{3}{2}c_i^{(1)}] \\
\forall i &= [1, \dots, n]
\end{aligned} \tag{3.25}$$

Arithmetical crossover utilizes a linear combination of parent chromosomes according to Equation 3.26 [141].

$$\begin{aligned}
O_i^{(1)} &= [\lambda c_i^{(1)} + (1 - \lambda)c_i^{(2)}] \\
O_i^{(2)} &= [\lambda c_i^{(2)} + (1 - \lambda)c_i^{(1)}] \\
\lambda &\in [0, 1] \quad (\text{uniform random}) \\
\forall i &= [1, \dots, n]
\end{aligned} \tag{3.26}$$

Geometric crossover is similar to arithmetical crossover, only it uses exponential scaling of

the parent genes as depicted in Equation 3.27 [142].

$$\begin{aligned}
O_i^{(1)} &= [(c_i^{(1)})^\omega + (c_i^{(2)})^{1-\omega}] \\
O_i^{(2)} &= [(c_i^{(2)})^\omega + (c_i^{(1)})^{1-\omega}] \\
\omega &\in [0, 1] \quad (\text{uniform random}) \\
\forall i &= [1, \dots, n]
\end{aligned} \tag{3.27}$$

simplex crossover (SPX) is introduced by Tsutsui et al. as a multiparent crossover operator that can produce multiple offspring [143]. Equation 3.28 depicts this process of expanding the simplex created by a desired number of parents, k_p , about the geometric center, G , by a user defined parameter, ζ , and creating new simplex points ($Y^{(j)}$) that define this new expanded simplex. From this point, a uniformly random sample is taken within this expanded simplex to generate a desired number of offspring (y).

$$\begin{aligned}
G &= \frac{1}{k_p} \sum_{j=1}^{k_p} P^{(j)} \\
Y^{(j)} &= (1 + \zeta)(G + P^{(j)}) \\
O^{(i)} &\in \text{uniform}(Y) \\
j &= [1, \dots, k_p] \\
i &= [1, \dots, y]
\end{aligned} \tag{3.28}$$

unimodal distribution crossover (UNDX) is a form of crossover that utilizes a normal distribution centered around the midpoint between two parents and adjusts the distribution based on a 3rd parent ($P^{(3)}$) [144]. Equation 3.29 illustrates the first step of finding the midpoint between $P^{(1)}$ and $P^{(2)}$ (G_m).

$$\vec{G}_m = \frac{1}{2}(P^{(1)} + P^{(2)}) \tag{3.29}$$

Next, the distance between $P^{(1)}$ and $P^{(2)}$ (d_p) is determined from Equation (3.30).

$$\vec{d}_p = (P^{(2)} - P^{(1)}) \quad (3.30)$$

From this point, the distance between $P^{(3)}$ and the line connecting $P^{(1)}$ and $P^{(2)}$, D_{p3} , is determined as outlined in Equation (3.31).

$$D_{p3} = |P^{(3)} - P^{(1)}| \times \sqrt{1 - \left(\frac{(P^{(3)} - P^{(1)})^T(P^{(1)} - P^{(2)})}{|P^{(3)} - P^{(1)}||P^{(1)} - P^{(2)}|} \right)^2} \quad (3.31)$$

Lastly, y offspring are created depending on to randomly chosen normally distributed parameters $\xi = N(0, \sigma_\xi^2)$ and $\eta_j = N(0, \sigma_\eta^2)$, as indicated in Equation (3.32) where $\vec{e} = \frac{\vec{d}_p}{|\vec{d}_p|}$

$$O^{(i)} = G + \xi d + \sum_{j=1}^{dim-1} \eta_j \vec{e}_j D_{p3} \quad (3.32)$$

$$i = [1, \dots, y]$$

Deb et al. introduce simulated binary crossover (SBX) which has become one of the more widely used crossover operators in real coded GAs [145], [146]. This operator utilizes two parents to create any desired number of offspring from the parents in accordance with Equation 3.33. Where u is determined from a uniform random number generator and η is a user-defined non-negative parameter that defines how concentrated these distributions are around each of the parents (larger values are more concentrated while smaller values are closer to uniform). Figure 3.9 illustrates this bimodal pdf.

$$\begin{aligned} O_i^{(1)} &= [(1 - \gamma)c_i^{(1)} + (1 + \gamma)c_i^{(2)}] \\ O_i^{(2)} &= [(1 + \gamma)c_i^{(1)} + (1 - \gamma)c_i^{(2)}] \\ \gamma(u) &= \begin{cases} (2u)^{\frac{1}{\eta+1}} & \text{if } u \leq \frac{1}{2} \\ [2(1 - u)]^{\frac{1}{\eta+1}} & \text{if } u > \frac{1}{2} \end{cases} \\ u &\in [0, 1] \quad (\text{uniform random}) \\ i &= [1, \dots, dim] \end{aligned} \quad (3.33)$$

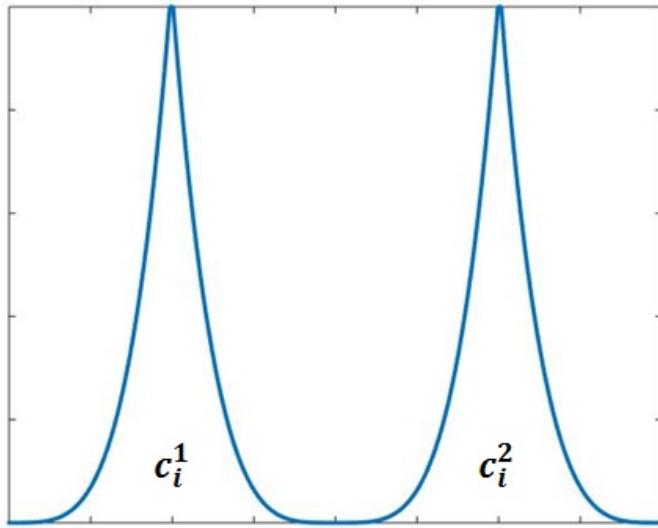


Figure 3.9. An example SBX probability density function

Fuzzy recombination is similar to SBX except for the use of triangular distributions around each of the parents [147]. This bimodal pdf is illustrated in Figure 3.10.

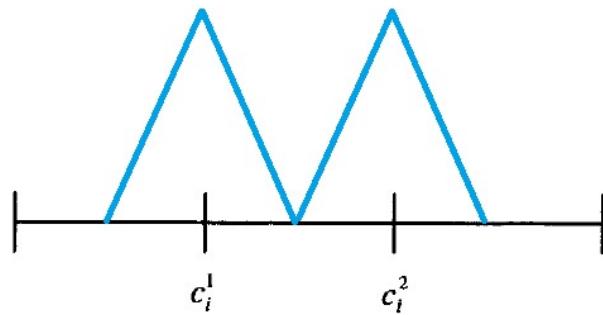


Figure 3.10. An example fuzzy recombination probability density function

It can be observed that the distribution-based crossover operators such as UNDX and SBX have a built in mutation characteristic, given that they have a low probability of producing an offspring that is significantly different from the parent vectors due to the tails of the distributions. However, the rest of the RCGA crossover operators that do not have this characteristic still require some type of mutation operator so that the RCGAs are able to escape local optima. In addition, the RCGAs that utilize the distribution crossover operators

may still achieve better performance with a mutation operator used in conjunction with the crossover operators.

3.15.2 Real Coded Genetic Algorithm Mutation Operators

Mutation operators can range from being fairly complex and dynamic to being very simple and static. Perhaps the most straightforward and intuitive RCGA mutation operator is simply a uniform random mutation where the mutated gene is chosen from a uniform distribution $O_i \in [LB_i, UB_i] \forall i = [1, \dots, n]$ with LB_i and UB_i being the upper and lower bounds in each dimension [141]. This is similar to bit-wise mutation for binary GAs with the difference being that an entire real-encoded parameter is mutated versus a single bit. Michalewicz also introduces a mutation operator that becomes less disruptive as the GA progresses that is referred to as non-uniform mutation [141]. This mutation operator is described further in Equation 3.34 where b is a user-defined parameter and g_{max} is the maximum number of generations.

$$O^{(i)} = \begin{cases} c_i + \delta(g, UB_i - c_i) & \text{if } \tau \leq \frac{1}{2} \\ c_i - \delta(g, c_i - LB_i) & \text{if } \tau > \frac{1}{2} \end{cases}$$

$$\delta(g, y) = y \left(1 - \alpha^{\left(1 - \frac{g}{g_{max}} \right)^b} \right) \quad (3.34)$$

$\tau \in [0, 1]$ (*Uniform Random*)

$\alpha \in [0, 1]$ (*Uniform Random*)

$i = [1, \dots, n]$

As can be imagined, a uniform distribution is not the only option to consider when developing a RCGA mutation operator. For certain problems, the practitioner may find that various distributions produce better results.

3.15.3 Real Coded Genetic Algorithm Theory

There have been several different approaches to describe the dynamics and convergence properties of RCGAs after they were first introduced in the late 1980s [80], [135], [137],

[139], [148]–[150]. Much of the initial work on RCGA theory was anchored on Holland’s Schema theory. Researchers defined RCGA schema and schema processing methodology in a way that can relate back to the original Schema Theory [137]. However, there have also been several authors that have taken slightly different approaches by modeling RCGAs as stochastic processes [149], [150] or developing predictions for expected progress [80].

Schema Theory Inspired Concepts

Goldberg was one of the first to investigate the theory associated with RCGAs by developing the schema theory in a way that fit these types of algorithms [137]. He was quick to point out the contradiction between practitioners who have seen superior results with RCGAs and the schema theory that suggests lower order alphabets should produce the best schema processing and thus algorithm performance. In order to describe this inconsistency, Goldberg develops the idea of virtual alphabets that allows schema theory concepts to become applicable to RCGAs. At the highest level, he postulates that RCGAs quickly pair down high-order alphabets into lower order virtual alphabets that then work in a similar fashion to binary GAs with the utilization of intrinsic parallelism and building block processing. More specifically, he illustrates that selection enables the RCGAs to quickly pair down the population to favor subsets or specific slices of the search region that can be obtained through crossover and selection. These slices then represent a virtual lower cardinality alphabet that the algorithm has essentially chosen for the user. This idea also illustrates that the ideas of deception and deceptive functions still manifests themselves, but in a slightly different way to that of binary GAs. It must be noted that this theory is developed with standard binary type crossover operators (e.g. single-point, multiple-point, and uniform) and not necessarily some of the operators that have been designed specifically for RCGAs described in the previous section.

In addition to Goldberg, several other authors have developed various ways of defining real-coded schema, making them more applicable to RCGAs [135], [139], [148]. These authors revisit schema theory and closely linked implicit parallelism concepts in the light of RCGAs. Wright introduces the idea of a real-coded schemata as one that restricts various parameters in a chromosome to sub-intervals of their feasible parameter space. Equation (3.35) illustrates this idea where a given parameter falls within a defined space $LB_i < x_i < UB_i$ and a sub-interval is defined by choosing real values α_i, β_i such that

$LB_i \leq \alpha_i < \beta_i \leq UB_i$ define a sub-interval between LB_i and UB_i in each dimension. Wright then re-defines the schema theorem using this definition for schema and operators that work in continuous spaces.

$$s = [(\alpha_1, \beta_1), \dots, (\alpha_n, \beta_n)] \in \prod_{i=1}^{i=n} [LB_i, UB_i] \quad (3.35)$$

Radcliffe generalizes the idea of a schema so that they can be applied to RCGAs and calls this generalization a forma. This in turn allows him to gain additional insight into various operators and allows for the investigation of implicit parallelism in the light of RCGAs [148]. Lastly, Eshelman et al. points out that the continuous variables that are used within RCGAs have limited computational precision internal to the computer that is the result of binary encoding [148]. Looking at it from this angle, these authors define an interval-schemata concept where the genetic operators operate on strings of integers instead of bit strings. Similarly to Wright, they define a schema as a set of ranges that a continuous parameter can take.

Expected Progress

Muhlenbein et al. use their Breeder GA, a cross between an ES and a RCGA, to develop the idea of expected progress as the result of mutation and selection with recombination [80]. As will be seen in the following chapters, it is difficult to avoid the discussion of ESs while investigating RCGAs given that they have many similarities. In fact, these authors are some of the first to leverage the abundance of theoretical work that has been developed in the ES arena. These authors develop an expected progress rate based an idea that is opposite to the schema theory in that they are interested in the creation of new, successful individuals rather than focusing on the disruption of existing schema as is the aspect of interest in the schema theory. They define a successful mutation with regard to expected progress as an improvement in Euclidean distance from the current point to the optimal point. This expected progress is calculated by integrating over the domain of successful mutations where the integrand is $progress(y) \times probability(y)$ where y is a successful mutation. Muhlenbein et al. also utilize a unimodal fitness function that is spherically symmetric. These techniques are ones that are typically utilized for ES theory and dynamic modeling of ESs [151]. Muhlenbein et al. find that the best mutation operator in terms of

expected progress on a spherical fitness function is a normally distributed mutation operator with an optimal standard deviation when compared to a uniform mutation operator and the mutation operator they use in their Breeder GA. In addition, Muhlenbein et al. also consider various selection and recombination operators but the focus of the analysis is to evaluate their specific Breeder GA.

Stochastic Process Model

Qi and Palmieri have perhaps conducted some of the most in-depth analysis in regard to modeling RCGAs by developing a discrete time stochastic process model for each of the operators [149], [150]. They assume a simplified RCGA with an infinite population, fitness proportional selection, an independent mutation operator. They initially develop a stochastic model to determine how the average population fitness evolves over time for a simple RCGA without crossover. The authors further develop their stochastic process model to develop several key results:

- A stochastic process model that defines how populations evolve in RCGAs for certain function classes
- Qi and Palmieri show monotonically increasing average population fitness values (on maximization problems) for a specific class of Lipschitz fitness functions using time-varying mutation rates
- Qi and Palmieri investigate convergence properties on a quadratic fitness function illustrating the role that population diversity plays in convergence speeds

In a second paper, Qi and Palmieri also increase the fidelity of this model to account for uniform crossover where only a single offspring vector is retained after each run of the crossover operator [150]. This work by Qi et al. essentially extends the Markov Process analysis described earlier to model RCGAs. Again, these conclusions are somewhat limited to specific classes of fitness functions and have similarities with the limitations that will be seen when ES theory is discussed in Chapter 5.

3.16 Useful Takeaways From GA and RCGA Literature Review

After conducting this extensive literature review on various forms of serial GAs, there are a number of concepts and lessons learned that will be applied in the following chapter. First of all, the basic structure of GAs including many of the standard operators are utilized throughout the experiments in Chapter 4. Coupled with these standard operators, the ranges of typical user-defined parameters such as p_{cross} and p_{mut} are leveraged to quickly find acceptable values for these parameters for a number of experiments in the following chapter. Next, lessons regarding binary GA encoding types are applied in the following chapter and reflective binary gray encoding is utilized to avoid negative affects associated with Hamming Cliffs. The theoretical discussion associated with both binary GAs and RCGAs help to develop an understanding of how various operator settings can impact the performance of these algorithms. Furthermore, the GAs theory and convergence studies are a critical element in a number of studies regarding population sizing and stopping criteria, both of which are directly leveraged in Chapter 4 to design effective algorithms. A number of the RCGA crossover operators outlined in the previous sections will be studied in detail and applied to solve optimal control problems. Next, the concepts and ideas associated with dynamic resolution and messy GAs are built upon to design new RCGA crossover operators and adapt existing ones to allow them to emulate some of the useful characteristics identified from these studies. While several of the concepts covered in the above sections, such as the schema theory and building block hypothesis are not directly applied, they are considered to be foundational to the study of GAs and are drawn upon to further understand the important characteristics of these types of algorithms. This study has helped to identify the current state of GA theoretical analysis and convergence proofs that provide the reader with a critical understanding of what can theoretically be expected in terms of algorithm performance. In addition, this literature survey has uncovered some of the state-of-the-art techniques utilized to address several of the key challenges associated with GAs. The last section of this literature review investigates the application of parallel computation as a tool to further enhance the performance of GAs.

3.17 Parallel Genetic Algorithm Architectures

Now that both binary GAs and RCGAs have been introduced and developed, the application of parallel processing on these types of algorithms can be discussed. It can be seen that GAs use very little problem information in optimizing a given problem of interest. This characteristic can be both disadvantageous and advantageous for several reasons. On one side, the algorithm becomes a stochastic search method that can take very large amounts of computation time to arrive at an acceptable solution. On the other side, this makes the algorithms very robust, able to handle discontinuous/non-differentiable search domains, and well suited for parallel computation. In fact, these algorithms can be run in parallel using several different techniques that can significantly reduce computation time, improve efficiency, increase algorithm robustness, and improve solution quality. It will be seen that several of the lessons regarding memory architectures and workload distribution investigated in Chapter 2 are directly applicable to several of these parallel GA architectures.

3.17.1 Master-Follower Genetic Algorithm

One of the most straightforward parallel GA constructs is the simple master-follower architecture [42]. Since GAs operate on sets of candidate solution vectors that all need to independently be evaluated against the fitness function of interest, this creates an almost perfectly parallel problem. In the master-follower construct, a single processor typically handles the less computationally expensive operations such as: selection, crossover, and mutation. Multiple processors are then used to split up the computational workload of evaluating the fitness function for subsets of the population as illustrated in Figure 3.11.

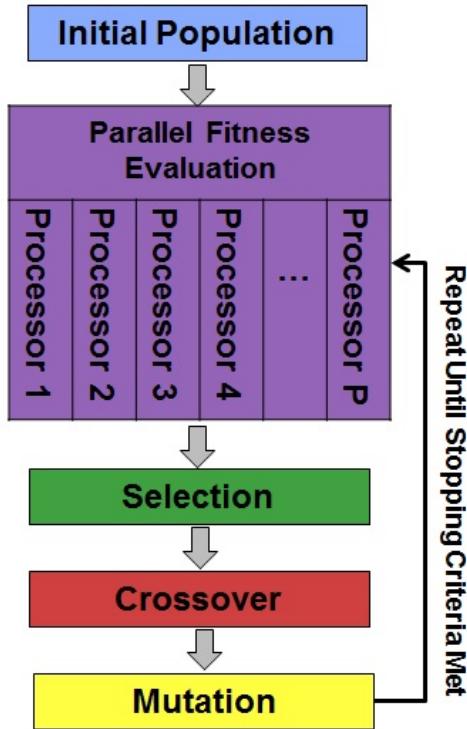


Figure 3.11. A master-follower parallel GA model

As an extreme case, a separate processor could be used to evaluate the fitness function for each individual chromosome. This, however, might not be an optimal scenario depending on the expense of fitness function evaluations. This master-follower implementation requires that all of the processors communicate their respective solutions back to the master processor during each iteration. This communication cost can prove to be fairly expensive and often results in an optimal number of processors that is less than the population size. Cantu-Paz and Goldberg illustrate that the computational time for this simple parallel GA architecture can be modeled using Equation (3.36) where T_p is the total processing time, P is the number of processors used, T_f is the time required for a single function evaluation, and N_{pop} is the GA population size [42]. It can be seen that this is very similar to Equation (2.21) studied in Chapter 2 with a few subtle differences. Equation (3.36) accounts for communication time, and does not account for separate serial portions of the code that cannot be parallelized as Equation (2.21) does. Given this, Equation (3.36) does not illustrate the fact that increasing P beyond N_{pop} will not generate any additional speedup, which is not entirely accurate

unless each individual function evaluation can be further parallelized.

$$T_p = PT_c + \frac{N_{pop}T_f}{P} \quad (3.36)$$

Furthermore, Cantu-Paz and Goldberg illustrate that this function can be differentiated with respect to the number of processors, P . This equation can then be set to zero and solved to find the optimal number of processors for a given communication cost, population size, and function evaluation time as depicted in Equation (3.37) [42]. Figure 3.12 shows a set of speedup curves for a simple master-follower GA that is created by using Equations (3.36) and Equation (3.37) with hypothetical values to illustrate how speedup is affected as the communication time is increased. Quite intuitively, it can be seen that as the communication cost decreases the optimal number of processors increases. Equation (3.37) also helps illustrate that as the function evaluations become more and more expensive, this cost will outweigh the cost of communication and thus the optimum number of processors will also increase.

$$P^* = \sqrt{\frac{N_{pop}T_f}{T_c}} \quad (3.37)$$

As was seen in Chapter 2 when investigating the parallel computation of RLV landing footprints on multicore CPUs, T_f and therefore T_p can also be dependent on the number of cores per processor when using multicore processors that share cache memory. While this is not incorporated in these simplistic models, it is a hardware-dependent factor that will also have an impact on the optimal number of processors that should be used for these types of algorithms. As was observed in Chapter 2, performance can be improved by not fully utilizing all of the cores on shared memory multicore CPUs, however, this is both hardware and problem dependent. With large enough population sizes and expensive enough fitness evaluations, the application of GPUs on these types of master-follower GAs has been shown to be successful under the right circumstances [152].

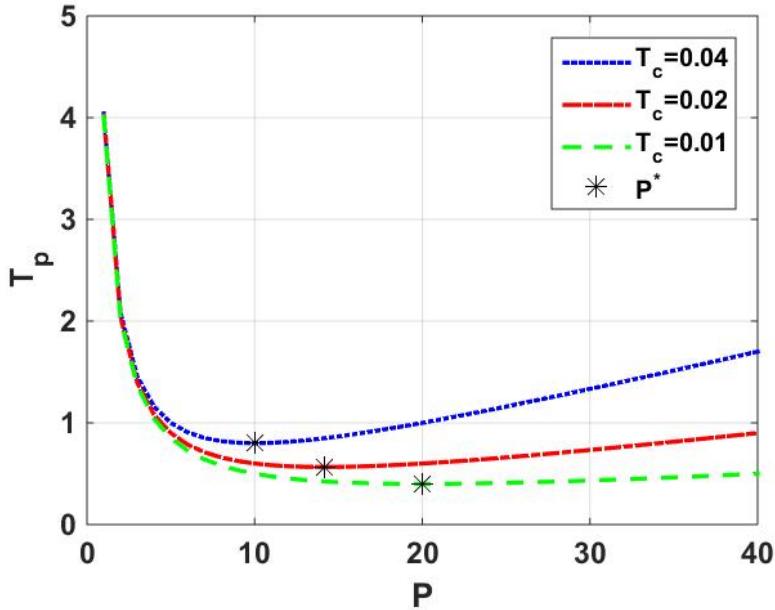


Figure 3.12. Theoretical speedup curves for a master-follower GA

3.17.2 Parallel Island Genetic Algorithm

A less communication-intensive parallel GA architecture is the parallel island GA which may also be referred to as: multiple deme, distributed, or coarse-grained [42]. This method allows a number of separate populations to evolve in a somewhat isolated fashion. This mirrors examples in biology where certain populations have evolved on islands in relative isolation. Both in nature and within the construct of these algorithms, migrations are permitted where isolated populations are able to share genetic information. Whereas the master-follower GA allows for increases in computational speed, the parallel island GA not only increases speed but fundamentally changes the evolution path and heuristics that occur within the algorithm [153]. In fact, the parallel island GA has the ability to achieve superior performance when compared to the sum of all of its individual components. [153] This can be a very desirable trait in parallel computing and is why this parallel GA architecture is the focus of research in the following chapter. While this dissertation focuses on the use of multi-processor, cluster computing devices, GPUs have also been used to run parallel island GAs in various forms [154]. It has been pointed out that coarse-grained or parallel island GAs in their pure form are not well-suited for GPU applications given that they require too

much communication relative to the number of required function evaluations [152].

In the case of the parallel island GA, this migration frequency is one of several user-defined parameters. Figure 3.13 illustrates a simple parallel island GA that uses a round-robin or ring message passing architecture. As one can imagine, the network geometry is yet another user-defined characteristic that can range from a simple round-robin idea, to one in which all islands communicate to all other islands during a migration. The number of chromosomes that are passed during each migration, the number of island populations, and the size of each island population are also user-defined parameters that can be tuned to adjust performance. These of course are all in addition to the standard user-defined parameters that exist within GAs described above.

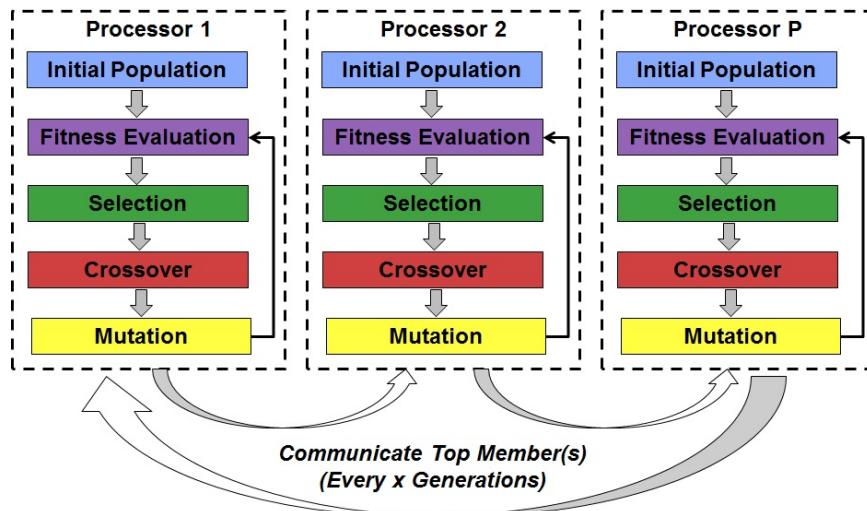


Figure 3.13. A round-robin parallel island GA model

Several studies have been done to look at how these parameters (e.g. migration rates, sizes, and frequencies) affect GA convergence efficiency [42], [155]–[159]. Many of these types of studies are empirical and not necessarily applicable to all types of problems and GAs. Cantu-Paz and Goldberg develop some theoretical results and models that are derived from Holland's Schema Theory and Goldberg's Building Block concept [42]. Unfortunately, while these models help to give insight into the relationships between number of island populations, migration rates, and population sizes, it is fairly difficult to directly apply them to real world problems where information about building blocks is not likely known. These ideas are even further complicated when trying to apply them to RCGAs.

There are a number of various migration methods where the number of chromosomes passed between populations, the way that they are selected, and the selection of chromosomes that they replace in the receiving population can all differ from one algorithm to the next. A fairly common method is where the chromosome with the best fitness is chosen as a single migrant from each population. Using the simple round-robin mapping, this chromosome that has been selected for migration replaces a random chromosome in the neighboring island population as long as it has an inferior fitness value than the migrant chromosome. Cantu-Paz has shown that it is important to have fitness-based selection of the migrants, but not necessarily so for the chromosomes that they replace [158]. Not having to search for the worst chromosome in a population to replace saves computation time, and does not significantly impact convergence rates or accuracy. Cantu-Paz refers to this method as a best-random migration scheme. In addition, it has been shown that smaller migration sizes seem to perform better and that migration frequency has a larger effect on convergence results than migration sizing or number of chromosomes passed during a migration [159]. While these general observations have been determined from empirical analysis, specific trends and optimal migration frequencies are considered to be problem-dependent. For these reasons, the following chapter focuses on single chromosome migrations and studies the migration frequency in accordance with what works best for the example problem under study. This dissertation does not claim this implementation is optimal in terms of the program architecture, or the computer hardware used, as even some of the most perfectly parallel problems can require a fair amount of analysis and planning to ensure this is the case, as observed in Chapter 2.

It must also be noted that the idea of parallel islands or distributed populations have also been used in serial where populations are separated into sub-populations on a serial GA [160]. In addition to this, shared memory applications have been done where there are overlapping subpopulations [161]. These algorithms use the overlapping regions as a method to allow results from one population to ripple through other populations and are sometimes referred to as fine-grained GAs when the number of sub-populations becomes large and their respective population size is fairly small. These specific types of island population GAs are better suited for GPU applications than the coarse-grained GAs initially described above [152].

3.17.3 Hybrid Parallel Genetic Algorithm

Hybrid parallel GAs are a combination of the previously described architectures where a master-follower application can be done within the construct of an island model parallel GA. [42] Figure 3.14 illustrates a simple model of how such an implementation could look. These algorithms have the potential to achieve the benefits from multiple methods by using the parallel island architectures at an increased speed [161]. However, this comes at the cost of requiring larger numbers of processors, more complicated workload allocations between processors, and the balance still exists where communication costs have to be considered when sizing the algorithms. This becomes increasingly important when memory coherency on multicore processors is considered [46].

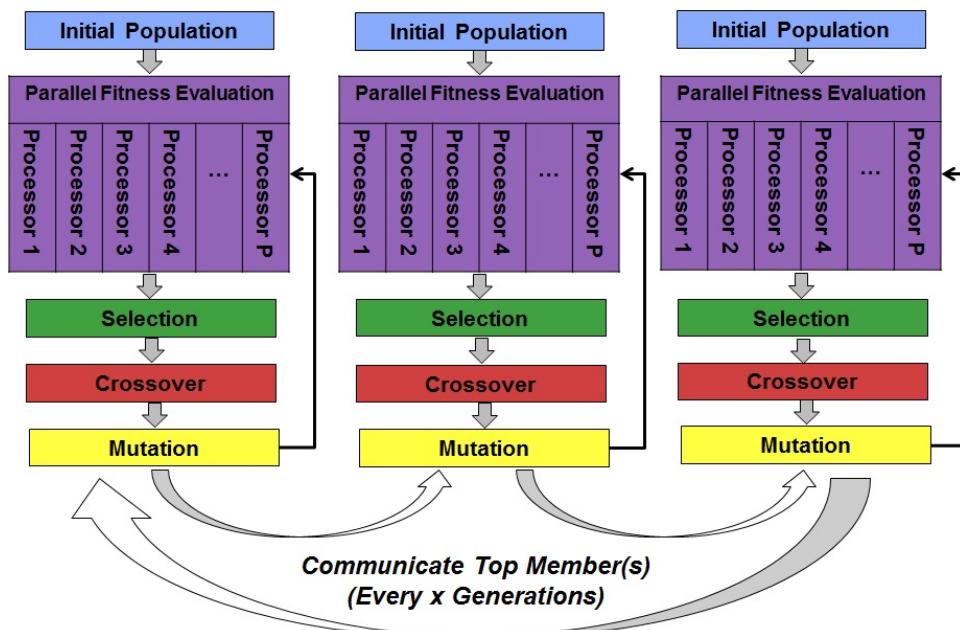


Figure 3.14. A hybrid parallel GA model

Hybrid parallel GAs are potentially one of the better parallel architectures for implementation on GPUs due to their requirement for hybrid parallel computing architectures that GPUs are setup to handle [152].

3.18 Conclusion

This chapter has developed the fundamental concepts and ideas associated with both binary GAs and RCGAs. The reader now has a general knowledge of how these algorithms work and how they are able to handle discontinuous, non-differentiable search spaces with very little information about the problem of interest. Furthermore, the reader now has a working knowledge of the fundamental theory associated with these types of evolutionary algorithms to include some effects associated with changing the various user-defined parameters. Lastly, the understanding of the various parallel implementations used to enhance the performance of these algorithms has also been introduced. The underlying knowledge associated with all of these concepts will be leveraged in the next chapter as these algorithms are studied through the lens of an optimal control problem. The ideas and topics outlined in this chapter are the results of a fairly extensive literature survey that will inform and inspire the new operators, methods, and algorithm designs developed in the next chapter.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 4: Parallel Genetic Algorithms and Optimal Control

4.1 Introduction

The first documented application of GAs in solving optimal control problems wasn't until 1990 [162]. Since this initial paper, there have been several other authors that have directly applied GAs to optimal control problems [163], [164]. Michalewicz illustrates that his RCGA performed better on simple, uniformly discretized, unconstrained optimal control problems than the MINOS solver in GAMS [163]. More specifically, he investigated a 1-D linear quadratic problem, a maximum distance/minimum effort push-cart problem, and a harvest maximization problem. Given these promising results, there have not been as many examples of direct applications of GAs to continuous variable optimal control problems as one might expect. There have been several authors that utilize GAs as guess generators that feed their solutions to deterministic methods as initial guesses to increase convergence probability [23], [165]–[168]. There have also been a growing number of papers where GAs are applied to hybrid optimal control problems, where there are both discrete and continuous decision variables [10], [23], [169], [170]. In these algorithms, the GA is typically applied to an outer loop problem that deals with optimizing the discrete decision variables, such as deciding which planetary bodies to visit in a multiple gravity assist interplanetary trajectory optimization problem. Deterministic methods such as sequential quadratic programming are then applied to the inner loop optimal control problem using a given set of discrete variables. However, Chilan and Conway use GAs for both the discrete variables in the outer loop, and as a guess generator for the continuous inner-loop dynamics in an interplanetary trajectory solver [23].

This chapter takes the initial studies by Michalewicz et al. [162], [163] one step further by using parallel GAs to directly solve an optimal control problem with continuous decision variables. As illustrated in Figure 4.1, this study is unique in that it investigates the use of binary and real-coded parallel island GAs as they apply to solving optimal control problems. It does this by analyzing the effectiveness of different encodings, operators, and constraint handling methods as these various GAs are run in parallel on cluster computing

devices. Several authors have studied the effects of varying parallel GA parameters in binary encoded GAs [42], [157] as well as in RCGAs [159]. In addition, authors have applied both RCGAs and binary GAs to continuous time optimal control problems (OCPs) [23], [163], [164], [166]. Lastly, researchers have conducted studies comparing the popular RCGAs operators [171]. However, a study that encompasses all of these trades and ideas as they apply to an OCP has not yet been conducted and is of specific interest to this dissertation. In addition, the study of how various forms of exact and inexact penalty functions perform through the lens of parallel GAs is also unique to this dissertation.

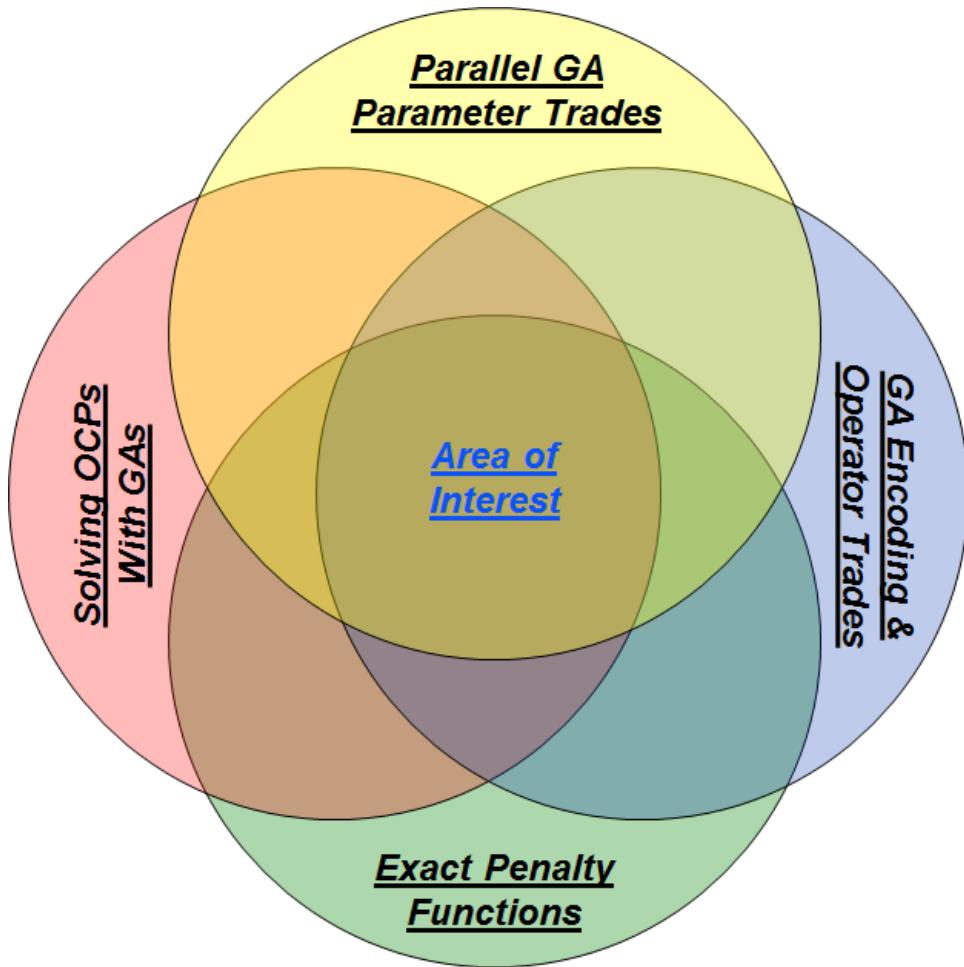


Figure 4.1. Venn diagram of parallel GA optimal control study

This chapter begins with an introduction to the lunar lander problem and the steps required to convert a continuous time OCP into a discrete NLP. From here, an introduction and

in-depth analysis of various penalty functions is conducted on two NLPs from literature that are deemed to be highly constrained and challenging. With this, the chapter then applies an exact penalty function to the lunar lander problem to convert it from a constrained NLP to an unconstrained one. Next, it introduces the different binary and real-coded GA types used for the experiments in this chapter. After this, the specific method of incorporating parallel processing into these algorithms is presented. The chapter then investigates several trades between algorithm types and parallel GA parameters. Lastly, the chapter concludes with a side-by-side comparison of the various parallel GA types that illustrates their effectiveness in solving the lunar lander problem. It becomes apparent that parallel computation leads to faster convergence (in terms of computation time) and more accurate results when compared to serial implementations. Furthermore, the improved performance achieved through parallel computing can result in an increasingly robust algorithm where perfect tuning of GA parameters becomes less critical in obtaining accurate solutions.

4.2 Lunar Lander Problem

This study utilizes a simplified lunar lander problem, depicted in Equation (4.1), to investigate GA performance on solving an example optimal control problem [31]. A solution for the minimum propellant thrust trajectory, $u(t)$, for a lunar lander under a given set of initial/final conditions, and bounds on both decision and state variables is desired. The state variables are altitude, $h(t)$, vertical velocity, $v(t)$, and spacecraft mass, $m(t)$. It can be seen that there are several constraints on both state and control variables that will require special constraint-handling techniques when working with evolutionary algorithms. The values in this problem have been scaled for computational efficiency and represent normalized units. The normalized constants for the problem are a gravitational acceleration constant, $c_g = 1.0$, and a specific impulse constant, $c_v = 2.349$ [31].

$$\bar{x}^T = [h(t) \in \mathbb{R}^n : 0.0 \leq h(t) \leq 20.0, v(t) \in \mathbb{R}^n : |v(t)| \leq 20.0, m(t) \in \mathbb{R}^n : 0.01 \leq m(t) \leq m_0]$$

$$\bar{u} = [u(t) \in \mathbb{R}^n : 0.0 \leq u(t) \leq 1.227]$$

$$\left\{ \begin{array}{l} \text{Minimize: } J[\bar{x}(\cdot), \bar{u}(\cdot), t_f] = \int_{t_0}^{t_f} \bar{u}(t) dt \\ \text{Subject To: } \dot{h} = v \\ \quad \dot{v} = -c_g + \frac{u}{m} \\ \quad \dot{m} = -\frac{u}{c_v} \\ \quad (h_0, v_0, m_0) = (1.0, -0.783, 1) \\ \quad (h_f, v_f) = (0.0, 0.0) \end{array} \right. \quad (4.1)$$

4.3 Direct Shooting

A direct shooting technique is used to convert the continuous lunar lander problem into a standard NLP. The shooting method essentially consists of applying the control trajectory to the dynamics, integrating either forward or backward in time (dependent on the problem), and then determining how well that given control trajectory satisfies the problem constraints and end-point/initial conditions [30]. A 3rd order strong stability preserving Runge-Kutta time integrator is used to integrate the lunar lander dynamic equations as depicted in Equation (4.2). This time integration technique allows for increased accuracy without adding too much additional complexity and computational effort, when compared to a simple Euler technique. In this chapter, uniform time discretization is utilized so that $\Delta t = \frac{t_f}{N_t}$. Where Δt is the time step size, t_f is the final time, and N_t is the number of time

nodes.

$$\begin{aligned}
\dot{\bar{x}} &= f(\bar{x}, \bar{u}, t) \\
\bar{x}(t_k) &= \bar{x}_k \\
\bar{x}_{k+1}^{(s1)} &= \bar{x}_k + (\Delta t)(f(\bar{x}_k)) \\
\bar{x}_{k+1}^{(s2)} &= \frac{3}{4}\bar{x}_k + \frac{1}{4}\bar{x}_{k+1}^{(s1)} + \frac{\Delta t}{4}f(\bar{x}_{k+1}^{(s1)}) \\
\bar{x}_{k+1}^{(s3)} &= \frac{1}{3}\bar{x}_k + \frac{2}{3}\bar{x}_{k+1}^{(s2)} + \frac{2\Delta t}{3}f(\bar{x}_{k+1}^{(s2)}) \\
k &= (0, \dots, N_t - 1)
\end{aligned} \tag{4.2}$$

This template in Equation (4.2) can be applied to the lunar lander problem dynamics and cost function in a way that converts the optimal control problem into a large NLP as described by Equation (4.3). This problem formulation is still subject to all of the original constraints at each node and end-point conditions depicted in Equation (4.1).

$$\begin{aligned}
\bar{x}^T &= [\bar{h} \in \mathbb{R}_t^N : 0.0 \leq \bar{h} \leq 20.0, \bar{v} \in \mathbb{R}_t^N : |\bar{v}| \leq 20.0, \bar{m} \in \mathbb{R}_t^N : 0.01 \leq \bar{m} \leq m_0] \\
\bar{u} &= [\bar{u} \in \mathbb{R}_t^N : 0.0 \leq \bar{u} \leq 1.227]
\end{aligned}$$

Minimize: $J[\bar{x}(.), \bar{u}(.), t_f] = \Delta t \sum_{i=0}^{N_t-1} u_i$
 Subject To: $(h_0, v_0, m_0) = (1.0, -0.783, 1)$
 RK3 Stage 1: $h_1^{(s1)} = h_0 + \Delta t v_0$
 $v_1^{(s1)} = v_0 + \Delta t \left(-c_g + \frac{u_0}{m_0} \right)$
 $m_1^{(s1)} = m_0 + \Delta t \left(-\frac{u_0}{c_v} \right)$
 RK3 Stage 2: $h_1^{(s2)} = \frac{3}{4}h_0 + \frac{1}{4}h_1^{(s1)} + \frac{\Delta t}{4}v_1^{(s1)}$
 $v_1^{(s2)} = \frac{3}{4}v_0 + \frac{1}{4}v_1^{(s1)} + \frac{\Delta t}{4} \left(-c_g + \frac{u_0}{m_1^{(s1)}} \right)$
 $m_1^{(s2)} = \frac{3}{4}m_0 + \frac{1}{4}m_1^{(s1)} + \frac{\Delta t}{4} \left(-\frac{u_0}{c_v} \right)$
 RK3 Stage 3: $h_1 = \frac{1}{3}h_0 + \frac{2}{3}h_1^{(s2)} + \frac{2\Delta t}{3}v_1^{(s2)}$
 $v_1 = \frac{1}{3}v_0 + \frac{2}{3}v_1^{(s2)} + \frac{2\Delta t}{3} \left(-c_g + \frac{u_0}{m_1^{(s2)}} \right)$
 $m_1 = \frac{1}{3}m_0 + \frac{2}{3}m_1^{(s2)} + \frac{2\Delta t}{3} \left(-\frac{u_0}{c_v} \right)$
 \vdots
 RK3 Stage 1: $h_{N_t-1}^{(s1)} = h_{N_t-2} + \Delta t v_{N_t-2}$
 $v_{N_t-1}^{(s1)} = v_{N_t-2} + \Delta t \left(-c_g + \frac{u_{N_t-2}}{m_{N_t-2}} \right)$
 $m_{N_t-1}^{(s1)} = m_{N_t-2} + \Delta t \left(-\frac{u_{N_t-2}}{c_v} \right)$
 RK3 Stage 2: $h_{N_t-1}^{(s2)} = \frac{3}{4}h_{N_t-2} + \frac{1}{4}h_f^{(s1)} + \frac{\Delta t}{4}v_f^{(s1)}$
 $v_{N_t-1}^{(s2)} = \frac{3}{4}v_{N_t-2} + \frac{1}{4}v_f^{(s1)} + \frac{\Delta t}{4} \left(-c_g + \frac{u_{N_t-2}}{m_f^{(s1)}} \right)$
 $m_{N_t-1}^{(s2)} = \frac{3}{4}m_{N_t-2} + \frac{1}{4}m_f^{(s1)} + \frac{\Delta t}{4} \left(-\frac{u_{N_t-2}}{c_v} \right)$
 RK3 Stage 3: $h_{N_t-1} = \frac{1}{3}h_{N_t-2} + \frac{2}{3}h_f^{(s2)} + \frac{2\Delta t}{3}v_f^{(s2)}$
 $v_{N_t-1} = \frac{1}{3}v_{N_t-2} + \frac{2}{3}v_f^{(s2)} + \frac{2\Delta t}{3} \left(-c_g + \frac{u_{N_t-2}}{m_f^{(s2)}} \right)$
 $m_{N_t-1} = \frac{1}{3}m_{N_t-2} + \frac{2}{3}m_f^{(s2)} + \frac{2\Delta t}{3} \left(-\frac{u_{N_t-2}}{c_v} \right)$
 $(h_{N_t-1}, v_{N_t-1}) = (h_f, v_f) = (0.0, 0.0)$

With this formulation, the decision variables that make up the GA trial vectors are the control variables ($\bar{u}(\cdot)$) in addition to the final time t_f . The number of equally spaced time nodes, N_t , is a user-defined parameter.

4.4 Constraint Handling

While GAs can be very effective in solving difficult optimization problems, the GA framework can only solve unconstrained problems. The standard operators such as crossover and mutation, which are used in most GAs, often result in infeasible solutions when solving constrained optimization problems. This results in algorithms that don't lend themselves well to constrained optimization without some additional techniques to handle constrained problems. Problems without any constraints are seldom (if ever) found when working with real-world applications due to the existence of physical limitations. Given the usefulness of GAs and the constrained nature of most optimization problems, there has been a significant amount of research and progress made with a number of constraint handling techniques that are applicable to GAs (Ref. Appendix B). The primary challenge associated with all of these constraint handling techniques is that their effectiveness can be dependent on both the algorithm type and type of problem being solved. This is still a very active research area, given that a robust, problem-independent GA constraint handling technique has yet to be truly developed and applied. For the purposes of this chapter, NLPs that can be written in the standard notation depicted in Equation (4.4) are used to illustrate the various constraint handling techniques.

$$\begin{aligned}
 & \text{minimize } f(\bar{x}) \\
 & \text{subject to:} \\
 & g_i(\bar{x}) \leq 0 \quad i = 1, \dots, l \\
 & h_k(\bar{x}) = 0 \quad k = 1, \dots, m
 \end{aligned} \tag{4.4}$$

where l is the number of inequality constraints, m is the number of equality constraints, and $\bar{x} \in \mathbb{R}^n$ is a n -dimensional solution vector. In addition to this, the search domain \mathbb{R}^n is often defined by bounds on the state variables $\bar{x}^U \leq \bar{x} \leq \bar{x}^L$. The feasible set \mathbb{F} is a subspace of \mathbb{R}^n that is created by the intersection of all constraints and state bounds $(g_i(\bar{x}) \cup h_k(\bar{x}) \quad \forall i, k \text{ s.t. } \bar{x} \in \mathbb{R}^n)$.

4.4.1 Penalty Functions

Penalty functions are the oldest and most widely used methods for constraint-handling in GAs [73]. They transform a constrained optimization problem into an unconstrained optimization problem through the construction of a fitness function, $F(\bar{x})$, that is typically a variation of the general form outlined in Equation (4.5). The new unconstrained problem can be created from a linear combination of the original objective function $f(\bar{x})$ and a penalized measure of constraint violation $p * P(\bar{x})$ where p is a penalty factor or weight. The measure of constraint violation $P(\bar{x})$ can be something as simple as the number of constraints violated, or some measure of the distance from an infeasible solution to the feasible region.

$$\text{minimize } F(\bar{x}) = f(\bar{x}) + p * P(\bar{x}) \quad (4.5)$$

While there are several different forms of penalty functions, this chapter uses the exact penalty function described in Equation (4.6). Di Pillo and Grippo prove a formal definition of exactness for the group of penalty functions described by Equation (4.6) [172]. These penalty functions are based on norms of both the inequality and equality constraints in the form shown in Equation (4.4). It must be noted that this form of an exact penalty function is only exact if the penalty parameter p is appropriately tuned. Exact penalty functions have several characteristics that make them very useful to GA applications. At a high level, an exact penalty function is one where the variables of the unconstrained problem formulation lie in the same space as the variables of the original constrained problem. In addition, the optimal solution to an exact penalty function is the same as the optimal solution to the original constrained problem [172].

$$F_q(\bar{x}) = f(\bar{x}) + p * \left[\sum_{i=1}^l (\max[0, g_i(\bar{x})])^q + \sum_{k=1}^m |h_k(\bar{x})|^q \right]^{\frac{1}{q}} \quad \infty > q \geq 1 \quad (4.6)$$

Equation (4.7) depicts the infinity norm which also falls within this class of exact penalty functions as defined by Di Pillo et al.

$$F_q(\bar{x}) = f(\bar{x}) + p * \left[\max \{(\max[0, g_1(\bar{x})]), \dots, \max[0, g_l(\bar{x})]), |h_1(\bar{x})|, \dots, |h_m(\bar{x})|\} \right] \\ \text{if } q = \infty \quad (4.7)$$

Due to the fact that penalty techniques have their roots in gradient-based search techniques, a great deal of effort has been spent on developing differentiable penalty functions. One

of the main luxuries of GAs is that they do not require differentiable functions, or for that matter, continuous problems. However, GA practitioners often use inexact penalty functions simply due to their abundance in literature and the optimization community. For completeness, a popular quadratic penalty function that is often used in GAs is described in Equation (4.8) [173].

$$F_q(\bar{x}) = f(\bar{x}) + p * \left[\sum_{i=1}^l (\max \{g_i(\bar{x}), 0\})^2 + \sum_{k=1}^m (h_k(\bar{x}))^2 \right] \quad (4.8)$$

Penalty Function Trade Study

A trade study is conducted to analyze the various traits of different penalty functions on two different constrained NLP test problems, hereby referred to as “problem 1” and “problem 2,” to generate an understanding of how they perform in conjunction with parallel GAs. Problem 1 is described in Equation (4.9) with two nonlinear constraints and two sets of box constraints or variable bounds [174]. This function’s nonlinear constraints create two overlapping circles with a small feasible region between them, as depicted in Figure 4.2. Note that these functions do not appear circular due given that the scale on the plot is skewed to emphasize the feasible region. This small feasible region makes the problem well-suited to test how several different penalty functions perform in terms of producing feasible and accurate solutions within the context of a parallel GA.

minimize:

$$f(\bar{x}) = (x_1 - 10)^3 + (x_2 - 20)^3$$

subject to:

$$-(x_1 - 5)^2 - (x_2 - 5)^2 + 100 \leq 0.0 \quad (4.9)$$

$$(x_1 - 6)^2 + (x_2 - 5)^2 - 82.81 \leq 0.0$$

$$x_1 \in [13, 100]$$

$$x_2 \in [0, 100]$$

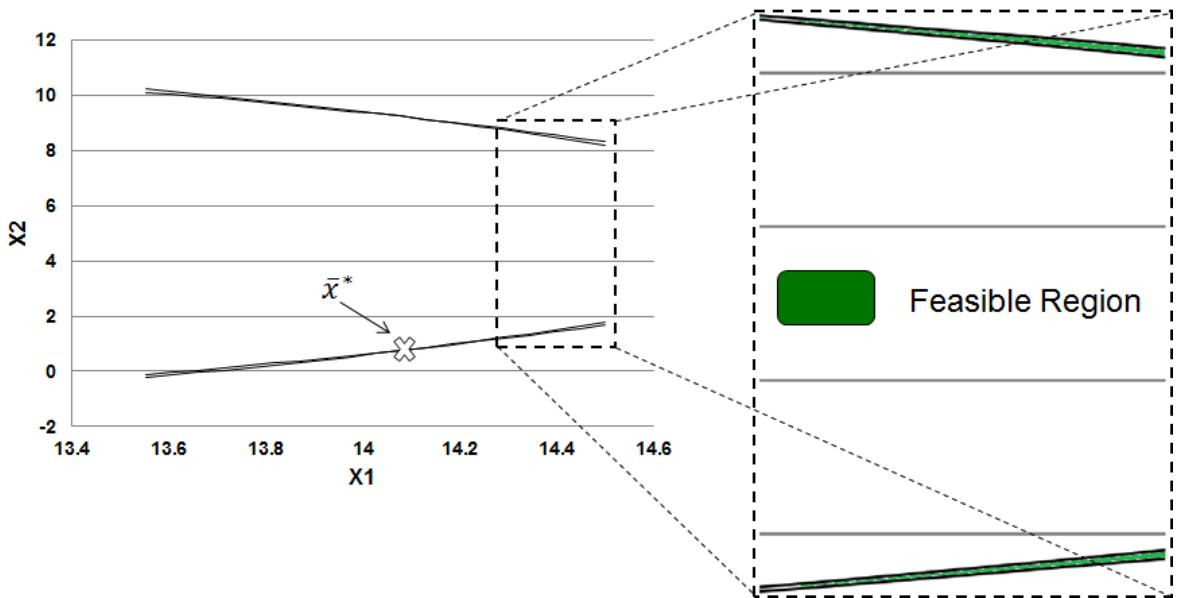


Figure 4.2. Illustration of feasible region for penalty function problem 1

Several different forms of penalty functions are applied to Equation (4.9) to produce several different unconstrained NLPs. Equation (4.10) illustrates an L1 norm penalty where $q = 1$ for Di Pillo's exact penalty function form (Equation (4.6)). It can be seen that the new problems become unconstrained NLPs where the box constraints on each variable are handled through the encoding/decoding process of the binary GA, as explained in the previous chapter.

$$\begin{aligned}
 F_{L1}(\bar{x}) &= (x_1 - 10)^3 + (x_2 - 20)^3 \\
 &+ p \left[\max \left\{ -(x_1 - 5)^2 - (x_2 - 5)^2 + 100, 0.0 \right\} \right] \\
 &+ p \left[\max \left\{ (x_1 - 6)^2 + (x_2 - 5)^2 - 82.81, 0.0 \right\} \right]
 \end{aligned} \tag{4.10}$$

Next, Equation (4.11) depicts an L2 norm penalty function where $q = 2$ using the form outlined in Equation (4.6).

$$\begin{aligned}
F_{L2}(\bar{x}) = & (x_1 - 10)^3 + (x_2 - 20)^3 \\
& + p \left[\left(\max \left\{ -(x_1 - 5)^2 - (x_2 - 5)^2 + 100, 0.0 \right\} \right)^2 \right. \\
& \left. + \left(\max \left\{ (x_1 - 6)^2 + (x_2 - 5)^2 - 82.81, 0.0 \right\} \right)^2 \right]^{\frac{1}{2}}
\end{aligned} \tag{4.11}$$

The next exact penalty function applied to problem 1 is the infinity norm penalty function where $q = \infty$ as illustrated in Equation (4.12).

$$\begin{aligned}
F_{Linf}(\bar{x}) = & (x_1 - 10)^3 + (x_2 - 20)^3 \\
& + p \left[\max \left\{ \left(\max \left\{ -(x_1 - 5)^2 - (x_2 - 5)^2 + 100, 0.0 \right\} \right) \right. \right. \\
& \left. \left. , \left(\max \left\{ (x_1 - 6)^2 + (x_2 - 5)^2 - 82.81, 0.0 \right\} \right) \right\} \right]
\end{aligned} \tag{4.12}$$

Lastly, the popular quadratic penalty function [173] is applied to test problem 1 in Equation (4.13) even though it does not fit the form of an exact penalty function developed by Di Pillo et al. [172] it is still of interest given that it is commonly used in GAs.

$$\begin{aligned}
F_{Quad}(\bar{x}) = & (x_1 - 10)^3 + (x_2 - 20)^3 \\
& + p \left[\max \left\{ \left(-(x_1 - 5)^2 - (x_2 - 5)^2 + 100, 0.0 \right) \right\}^2 \right. \\
& \left. + \left(\max \left\{ (x_1 - 6)^2 + (x_2 - 5)^2 - 82.81, 0.0 \right\} \right)^2 \right]
\end{aligned} \tag{4.13}$$

Using a simple binary-encoded GA with tournament selection and multi-point crossover operator, a series of experiments is conducted to evaluate several types of penalty functions. The second column of Table 4.1 summarizes the specific GA settings that are used to produce the results observed for test problem 1.

	Problem 1	Problem 2	Constraint Test (Parallel)
Encoding	Binary	Binary	Binary
Population	30	30	30
Procs.	10	10	(10)
Mig./MaxGen	$10/10^4$ Gen.	$10/10^4$ Gen.	(5/500 Gen.)
Migration Type	Best Chrom.	Best Chrom.	(Best Chrom.)
Crossover	2-Point	5-Point	1-Point
Resolution	32 bits/var	32 bits/var	20 bits/var
Selection	Tourn. (pool=2)	Tourn. (pool=2)	Roulette
p_{mut}	0.01	0.01	0.01
p_{cross}	0.6	0.6	0.3

Table 4.1. Attributes of parallel island GAs used for penalty experiments

Given that problem 1 is only 2-D, it is fairly easy to visualize where the solutions lie in the search space. Figure 4.3 illustrates the results obtained after 10^4 generations using the simple, binary-encoded GA described in the second column of Table 4.1. This simulation is run 1,000 times for each form of penalty function and the final solution found for each GA run is plotted. In general, it can be seen that the feasibility of solutions to this problem improve as the penalty factor is increased. In fact, Di Pillo et al. state that for a penalty function to be exact, the penalty parameter, p , has to be sufficiently large [172]. Figure 4.3a illustrates the case where the penalty parameter is not large enough. In this case, some solutions that are infeasible in the original problem formulation result in better objective function values than feasible solutions, indicating that the penalties being applied to these solutions are not large enough. On the other hand Figure 4.3b illustrates the effect of increasing p to where it becomes difficult, or even impossible for an infeasible solution to obtain a better fitness value than a feasible one, due to the large penalty weight.

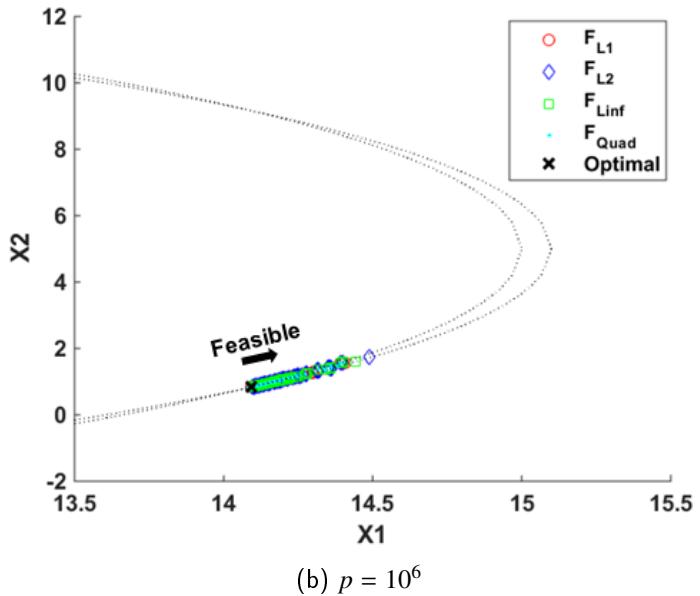
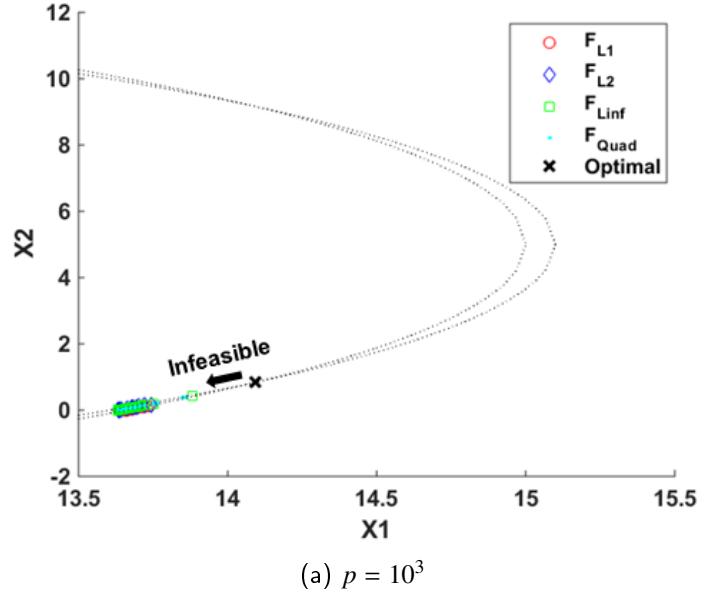


Figure 4.3. Effects of penalty parameter value on solution feasibility for problem 1

Unfortunately, this trade is not as black and white with GAs as one might think. In fact, optimization problems often have optimal solutions that lie on constraint boundaries where

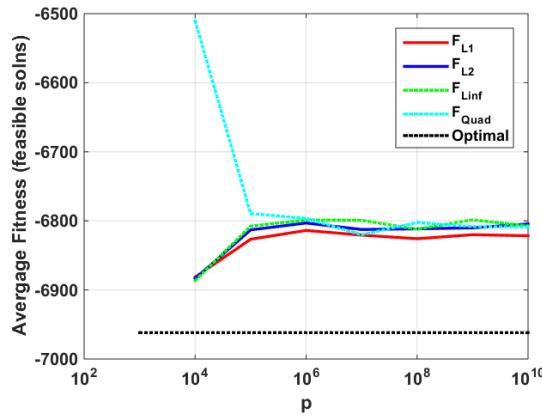
the best objective function value is obtained by pushing a given system parameter to one (or more) of its limits. Since GAs explore the search space using recombination operators such as crossover, it is difficult for the algorithms to search the boundaries of the solution space without any infeasible solutions in the population [175]. Understanding this, the penalty parameters need to be tuned in a way that allows some penalized solutions (solutions that are infeasible in the original constrained NLP) to remain in the population so that the GA is better able to search along constraint boundaries. Figure 4.4 provides more detail on the fitness and feasibility results associated with each penalty function type and penalty parameter value for problem 1. More specifically Figure 4.4a illustrates the average fitness value of the feasible solutions found over the 1,000 GA runs for each penalty function type and parameter value considered. It must be noted that none of the methods found feasible solutions for the case $p = 10^3$ and therefore no average fitness value is shown for this instance on the plots. It can be seen that the best average fitness values seem to occur where the penalty parameter is just on the cusp of producing both infeasible and feasible solutions ($p = 10^4$). Later in this chapter, an adaptive penalty technique is used to appropriately tune the penalty parameter for the lunar lander problem. With the quadratic penalty function not fitting into the same family of penalty functions as the others, it is not too surprising that it appears to behave differently than the exact penalty functions.

Figures 4.4b and c illustrate two slightly different ways of measuring the infeasibility of resulting solutions for each case. Figure 4.4b simply looks at the total number of infeasible solutions, or ones that violate any of the constraints of the original NLP for the 1,000 GA runs for each penalty function type and penalty parameter setting. The solution found, \bar{x}_{trial} , is the best member of the population after a GA run. Next Figure 4.4c provides a little more detail in terms of how infeasible each of the solutions are. Referring to Equation (4.14) a simple average measure of infeasibility, $infeas_{avg}$, is determined for each of the solutions that violate one or more constraints $g_i(\bar{x}) \leq 0$ where $\#_{constraints}$ is the number of constraints in the NLP and $\#_{infeasible}$ is the number of infeasible solutions encountered for each set of trials. Furthermore, the function $w_i()$ returns zero if the current solution meets

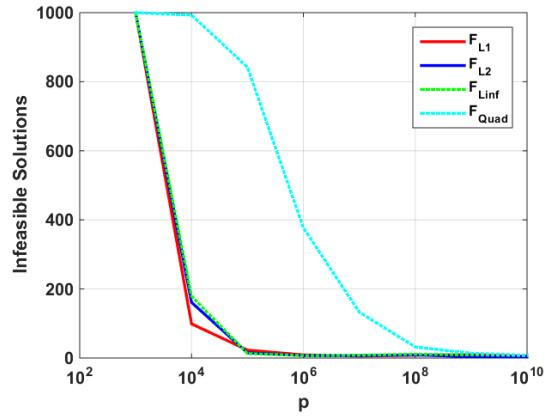
the i th constraint, and returns a one if it does not.

$$infeas_{avg} = \frac{\sum_{trial=1}^{\# trials} \sum_i^{\# constraints} w_i(\bar{x}_{trial})}{\# infeasible}$$

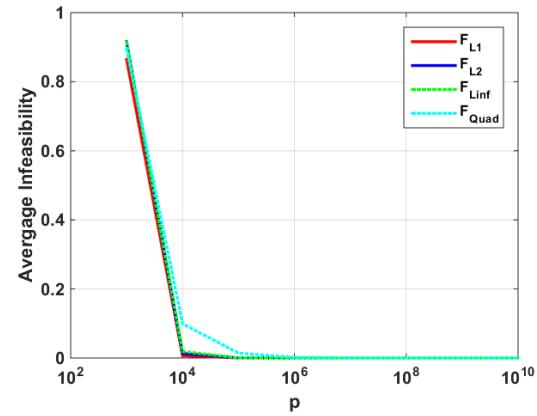
$$w_i(\bar{x}_{trial}) = \begin{cases} 0 & \text{if } g_i(\bar{x}_{trial}) \leq 0 \\ g_i(\bar{x}_{trial}) & \text{if } g_i(\bar{x}_{trial}) > 0 \end{cases} \quad (4.14)$$



(a) Average fitness of feasible solutions



(b) Number of infeasible solutions



(c) Average infeasibility

Figure 4.4. Average effects of penalty function type and parameter value for problem 1

Equation (4.15) depicts the second test problem [174] (problem 2) with a larger number of

nonlinear constraints and a higher-dimensional search space. This problem is used to further investigate how the various types of penalty functions perform as penalty parameters are adjusted. The attributes used on the parallel island binary GA are listed in the 3rd column of Table 4.1.

minimize:

$$f(\bar{x}) = 5.3578547x_3^2 + 0.8356891x_1x_5 + 37.293239x_1 - 40792.141$$

subject to:

$$\begin{aligned} 0.0022053x_3x_5 - 85.334407 - 0.0056858x_2x_5 - 0.0006262x_1x_4 &\leq 0.0 \\ 85.334407 + 0.0056858x_2x_5 + 0.0006262x_1x_4 - 0.0022053x_3x_5 - 92 &\leq 0.0 \\ 90 - 80.51249 - 0.0071317x_2x_5 - 0.0029955x_1x_2 - 0.0021813x_3^2 &\leq 0.0 \\ 80.51249 + 0.0071317x_2x_5 + 0.0029955x_1x_2 + 0.0021813x_3^2 - 110 &\leq 0.0 \end{aligned} \tag{4.15}$$

$$\begin{aligned} 20 - 9.300961 - 0.0047026x_3x_5 - 0.0012547x_1x_3 - 0.0019085x_3x_4 &\leq 0.0 \\ 9.300961 + 0.0047026x_3x_5 + 0.0012547x_1x_3 + 0.0019085x_3x_4 - 25 &\leq 0.0 \\ x_1 &\in [78, 102] \\ x_2 &\in [33, 45] \\ (x_3, x_4, x_5) &\in [27, 45] \end{aligned}$$

For brevity, the various forms of penalty functions as applied to test problem 2 along with their associated NLPs are not listed in this section. However, the same four types of penalty functions ($F_{L1}(\bar{x})$, $F_{L2}(\bar{x})$, $F_{Linf}(\bar{x})$, and $F_{Quad}(\bar{x})$) used for test problem 1 are also used for test problem 2. Figure 4.5 illustrates similar trends as before, where the GA actually achieves better solutions when p is tuned in a way that forces solutions to just be on the verge of feasibility so that constraint boundaries can be explored. In fact, Figure 4.5a illustrates that both the $F_{L1}(\bar{x})$ and $F_{L2}(\bar{x})$ are able to produce optimal solutions to this notoriously difficult problem when $p = 10^3$. Looking at Figure 4.5c and d helps the reader verify that only a small fraction of the 1,000 trials produced feasible solutions for the $F_{L1}(\bar{x})$ and $F_{L2}(\bar{x})$ penalty functions, however, the solutions that were feasible were also optimal at this penalty parameter setting. This helps further illustrate the trade that occurs between feasibility and optimality when tuning penalty parameters. In addition, comparing the results between the

2 test problems (Figures 4.4 and 4.5) helps the reader get an understanding for how these penalty parameters can be problem dependent.

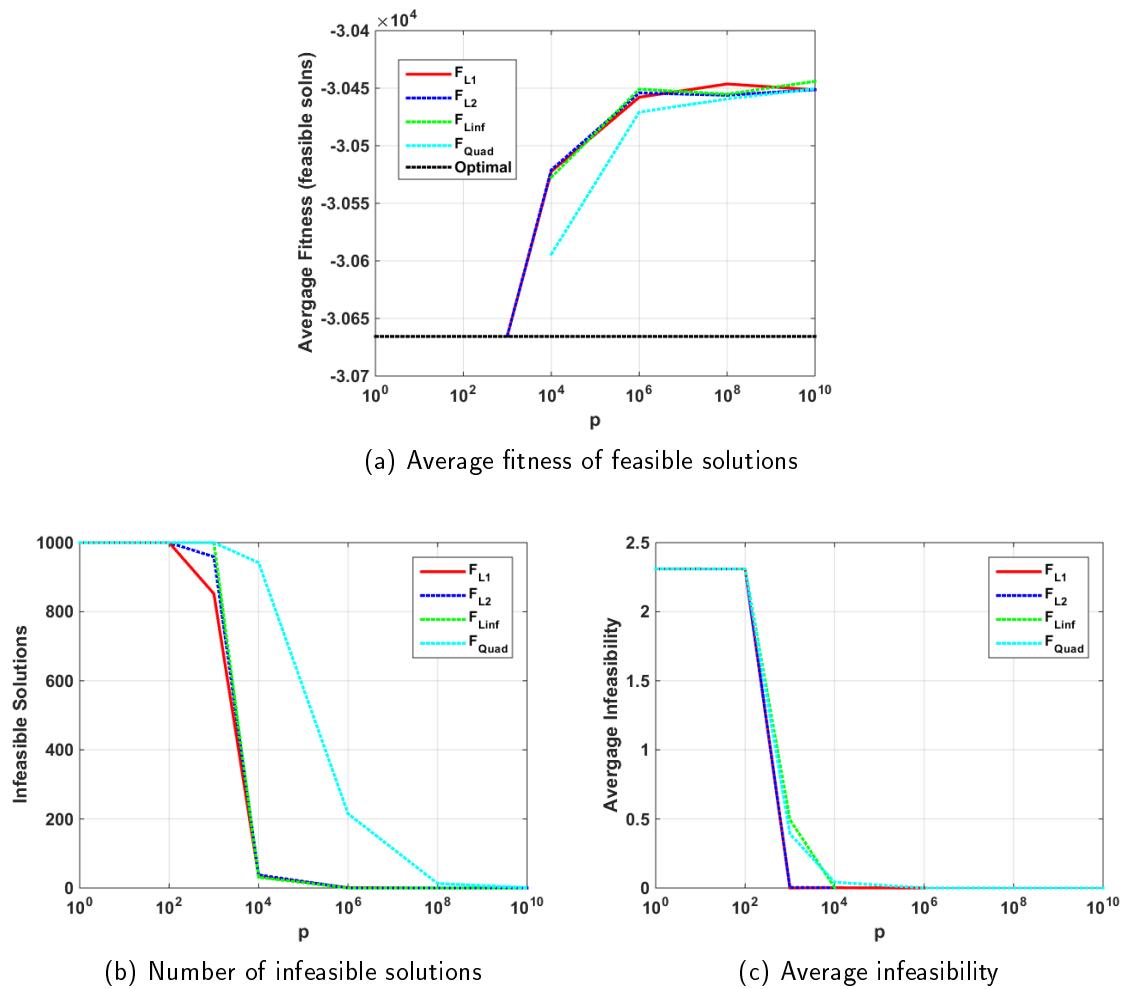


Figure 4.5. Average effects of penalty function type and parameter value for problem 2

It can be seen that the above example problems do not contain any equality constraints. However, it is important to analyze how equality constraints can affect the solution quality obtained using these algorithms. For this reason, an experiment is conducted to analyze how well GAs are able to handle equality constraints when using exact penalty functions. More specifically, several authors have shown that GAs produce better results when equality

constraints are converted into sets of inequality constraints [176]. Equation (4.16) and Equation (4.17) illustrate two slightly different NLP formulations with the same optimal solution to both. It can be observed that Equation (4.16) is a simple parabolic objective function with the constraint $x_1 - x_2 + 1 = 0$ being enforced through an exact penalty function. Next, Equation (4.17) illustrates the same simple parabolic objective function with an inequality constraint $x_1 - x_2 + 1 \leq 0$ being enforced using an exact penalty function. It must be noted that these two formulations are not the same problem, however they do have the same optimal solution. An experiment is conducted to evaluate how the L1 exact penalty function handles each formulation in terms of proximity to the optimal solution. This experiment was originally conducted and documented by Seywald et al. [176] in serial and is recreated with the simple parallel island binary GA that has been used to evaluate the above penalty functions as outlined in column four of Table 4.1. Figure 4.6 illustrates the effects of switching from an equality constrained problem, Equation (4.16), to an inequality constrained problem, Equation (4.17). This experiment also investigates how a simple ten processor parallel island GA can improve the quality of solutions in both cases.

$$F_{equality}(\bar{x}) = x_1^2 + x_2^2 + 10^6 |x_1 - x_2 + 1| \quad (4.16)$$

$$F_{inequality}(\bar{x}) = x_1^2 + x_2^2 + 10^6 \max \{x_1 - x_2 + 1, 0.0\} \quad (4.17)$$

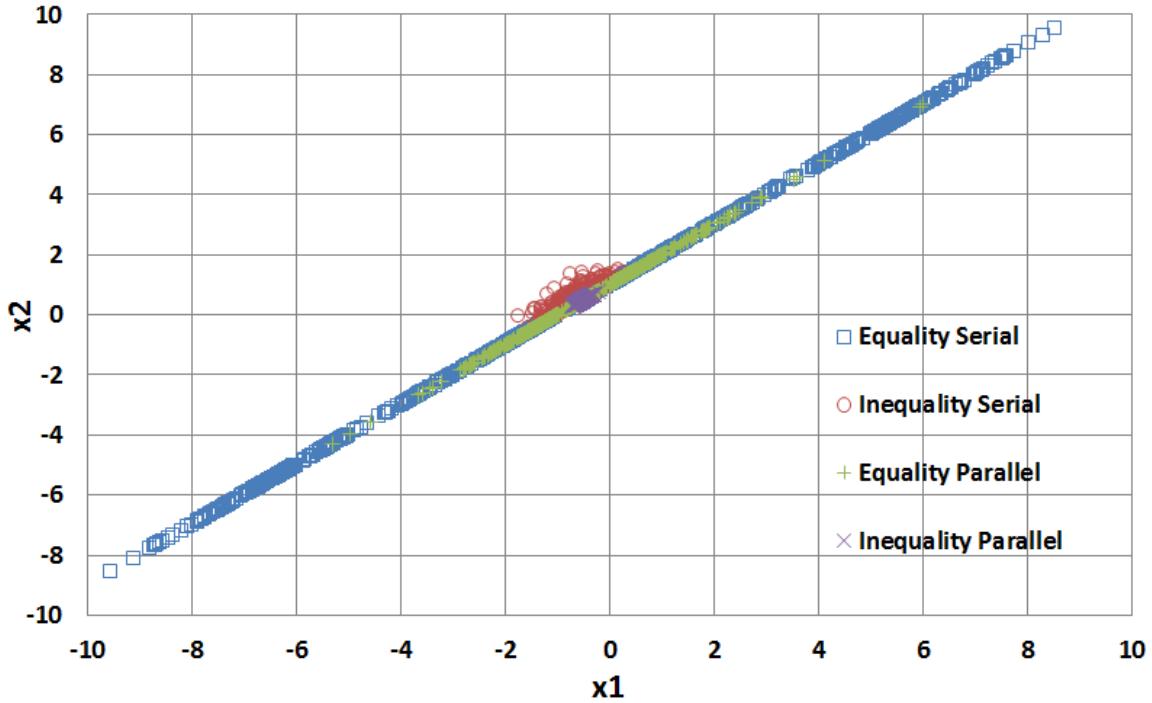


Figure 4.6. Illustration of equality formulation vs. inequality formulation for parallel and serial GAs

In concurrence with the findings of Seywald et al., the results of this experiment indicate that the algorithms are able to achieve better solutions on average (closer proximity to the optimal) when equality constraints are converted to inequality constraints. For this reason, a slightly modified version of this technique is used to convert equality constraints on the lunar lander problem in the hope of achieving better results. When converting equality constraints to inequality constraints, care must be taken to ensure that the new problem formulation does not violate the intent and requirements of the original problem.

Application of Penalty Functions to The Lunar Lander OCP

The results above indicate that exact penalty functions perform better than the quadratic form when using parallel GAs. In addition, the L1 penalty function seems to produce slightly better results than the other exact penalty functions evaluated. With this in mind, a simple L1 norm ($q = 1$) penalty function will be used in the GAs for the remainder of

this chapter. Lastly, the above results indicate that better results are achieved by converting equality constraints into inequality constraints, when possible. Given this, a conversion method outlined in Equation (4.18) is applied to the lunar lander problem. This method, where tol is a very small user-defined tolerance, has been shown to be more effective than using equality constraints [176]. The value of acceptable error, tol should be set to a value that is acceptable to the user and application of interest. It should be pointed out that there will always be some level of error when using numerical computation due to machine precision levels associated with the number of bits allocated to given variables.

$$h(x) = 0 \rightarrow -tol \leq h(x) \leq tol \quad (4.18)$$

After applying Equation (4.18) to the lunar lander NLP given in Equation (4.3), the problems are effectively converted into unconstrained optimization problems that the GA variations can attempt to solve with the new objective function depicted in Equation (4.19). The constraint on the control variable \bar{u} is not included in this formulation because it is a decision variable and will be handled with scaling and other methods as explained later. Using altitude at each node h_i as an example, an upper bound term in Equation (4.19) enforced by $\max(0.0, h_i - UB_i)$ and a lower bound is enforced by $\max(0.0, LB_i - h_i)$. It can also be seen that the end-point equality constraints have been converted to inequalities using a tolerance as discussed above.

$$\begin{aligned} \bar{x}^T &= [\bar{h} \in \mathbb{R}_t^N, \bar{v} \in \mathbb{R}_t^N, \bar{m} \in \mathbb{R}_t^N] \quad \bar{u} = [\bar{u} \in \mathbb{R}_t^N] \\ F(\bar{x}, \bar{u}, t_f) &= \Delta t \sum_{i=0}^{N_t-1} u_i + p \sum_{i=0}^{N_t-1} [\max(0.0, 0.0 - h_i) + \max(0.0, h_i - 20.0)] \\ &\quad + p \sum_{i=0}^{N_t-1} [\max(0.0, -20.0 - v_i) + \max(0.0, v_i - 20.0)] \quad (4.19) \\ &\quad + p \sum_{i=0}^{N_t-1} [\max(0.0, 0.0 - m_i) + \max(0.0, m_i - m_0)] \\ &\quad + p[\max(0.0, -tol - h_f) + \max(0.0, h_f - tol)] \\ &\quad + p[\max(0.0, -tol - v_f) + \max(0.0, v_f - tol)] \end{aligned}$$

4.4.2 Adaptive Penalty

This chapter also investigates a simple adaptive penalty where the value of the penalty parameter, p , is adjusted real-time to ensure that there is proper scaling between the penalty term, $p * P(\bar{x})$, and the objective function term, $f(\bar{x})$. This makes the GAs more robust and efficient given that the user does not have to tune fixed penalty parameters based on trial and error. One of the better performing and least complicated adaptive penalty methods was introduced by Hadj-Alouane and Bean [175]. This simple adaptive penalty scheme uses information about the best solution found in each population to adjust the penalty parameter in real-time. Their algorithm is used to initially tune p for the GAs in this chapter and is further outlined in Equation (4.20). In this adaptive scheme, “case 1” is where the best solution found for each iteration has not been penalized for a user-defined number of iterations, g_{adapt} . This is an indication that the penalty parameter is too large and needs to be decreased. The other scenario is “case 2” where the best solution found in each population is penalized every generation for g_{adapt} generations. This indicates that the penalty parameter is too small and needs to be increased. The goal is to tune the penalty parameter so that some of the best solutions are being penalized and some are not. This ensures that the objective function term, $f(\bar{x})$, is approximately the same magnitude as the penalty term, $p * P(\bar{x})$ which helps promote search along the constraint boundaries [175].

$$p_{(g+1)} = \begin{cases} \frac{p_{(g)}}{\beta_1} & \text{if case 1} \\ p_{(g)} * \beta_2 & \text{if case 2} \\ p_{(g)} & \text{else} \end{cases}$$

Case 1: $\bar{x}_{population\ best}$ is never penalized over g_{adapt} generations. (4.20)

Case 2: $\bar{x}_{population\ best}$ is penalized every generation for g_{adapt} generations.

$$(\beta_1, \beta_2) > 1$$

$$\beta_1 > \beta_2$$

It must be noted that the tuning rate for decreasing the penalty parameter, β_1 , and the rate for increasing the penalty parameter, β_2 are both user-defined parameters. These parameters essentially establish how fine (or coarse) the tuning of the penalty parameter is. As an example, β values that are closer to 1.0 result in a much finer tuning of the penalty parameter which comes at the cost of slower convergence to a given penalty parameter. On

the other hand, larger β values will result in faster convergence to a penalty parameter, but will also lead to a less-finely tuned penalty parameter that could lead to less accurate GA solutions to the problem. This is a trade that the user has to make in determining acceptable values for these parameters and can be done through empirical investigation.

4.5 An Overview of Genetic Algorithms Used

With an unconstrained form of the lunar lander problem generated through the application of penalty methods described above, the GAs used to solve the lunar lander problem are now described in more detail. Several new ideas including dynamic, Gaussian-based crossover operators that are unique to this dissertation are developed and evaluated in this section. Several of these new techniques are inspired by the dynamic resolution GAs discussed in Section 3.14.2. More specifically, new dynamic Gaussian crossover operators are developed to offer a dynamic resolution aspect to RCGAs. The hope is that some of the benefits observed by Andre et al. for binary dynamic resolution GAs [128] will also be found with these new techniques. In addition, these new crossover operators are designed to incorporate the fitness values associated with each of the parents in the creation of new offspring. As seen in Chapter 3, this is a practice that is not typically done with RCGA crossover operators.

4.5.1 Binary Reflected Gray Encoded GA

A fairly standard binary GA is utilized as one of the more conventional algorithms for comparison with some of the RCGAs investigated in this study. Given the results and recommendations of the encoding studies outlined in the previous chapter [82], [83], reflected binary gray encoding is utilized in the binary GA for these studies. With this, each problem variable is encoded using 64 bits. In addition, single-point crossover is utilized with fairly large populations as recommended by De Jong et al. [85]. Simple, bit-wise mutation is applied to this algorithm as well with a fairly low mutation rate, $p_{mut} = 0.001$. While this is on the lower end of the spectrum for standard ranges of crossover probabilities that practitioners typically use for binary GAs [86], it is coupled with an additional mutation reset mechanism which will prove to produce good results.

The reset mechanism utilized in this binary GA simply sets the mutation probability to a

high value (e.g. $p_{mut} = 0.5$) for a single generation when the average population fitness value stagnates or does not change beyond a user-defined tolerance over a set number of generations. This aspect is added to the standard binary GA given that it has been shown to enhance performance in several of the studies investigated in Chapter 3 [128], [176]. For the experiments later in this chapter, this tolerance is set to 10^{-6} and the algorithm checks for stagnation every 10 generations. These values are problem/algorithm dependent and have been adjusted specifically for the lunar lander problem and algorithm settings used in this chapter. This additional mutation step reshuffles the current population whenever the GA seems to have converged so that it is able to explore different areas of the search domain. This modification proved to be beneficial based on empirical results obtained during the parameter tuning process. In addition to this reset method, tournament selection with a fairly large pool size is utilized to quickly exploit the search space represented by a given population. This tournament selection operator considers both parents and offspring with replacement so that it is choosing from a set of solution vectors that is twice the population size. In addition, elitism is enforced so that the best solution found is always represented in the current population. Lastly, the stopping criteria for all GAs utilized in the remainder of this chapter is simply a maximum number of generations. The specific values and remaining user-defined parameters are further defined in Table 4.2.

4.5.2 RCGAs

The RCGAs utilized in this chapter are similar in many ways to the binary GA described in the previous section. They utilize an identical tournament selection operator and very similar mutation method that operates on the real variables. The RCGAs all use a mutation operator that steps through each variable of each candidate solution vector and use uniform random distributions to change the value of a given variable within the allowable search space. The probability of this mutation occurring, p_{mut} , is again defined by the user as is done with the binary GA above. However, these algorithms do not use the mutation resets that the binary GA does. Instead, they rely on their crossover operators, which in many cases also have a random mutation aspect built into them. The more obvious difference is that each decision variable c_i is a double-precision real number where parent chromosomes can be represented as $P^{(1)} = [c_1^{(1)}, c_2^{(1)}, \dots, c_{dim}^{(1)}]$ and $P^{(2)} = [c_1^{(2)}, c_2^{(2)}, \dots, c_{dim}^{(2)}]$ and resulting offspring are represented by $O^{(1)}$ and $O^{(2)}$. The following crossover operators are either new, or have been modified from their original form to specifically address the challenges

associated with the lunar lander problem introduced in this chapter.

Arithmetic Crossover

Arithmetic crossover, as introduced earlier, utilizes a linear combination of parent chromosomes according to Equation 4.21 [163]. Box constraints on decision variables are naturally enforced in binary GAs through the encoding/decoding process, however, this is not necessarily the case with RCGAs. In the RCGA used to obtain the results below, the arithmetic crossover technique is further modified by the addition of a simple repair method, $R(O_i)$, that is used to enforce upper bounds, UB , and lower bounds, LB , on decision variables. Repair methods, such as this, are not new and have been applied in various GAs to handle simple box constraints in the past [177], [178]. This simple repair method sets any variable in violation of an upper or lower bound equal to the bound that it has violated. It must be noted that the repair method is only utilized for the simple box constraints on the control variables, \bar{u} , in lieu of a penalty function to more closely parallel a binary encoded GA. Penalty functions are still used for all of the constraints on state variables in addition to initial and final conditions.

$$\begin{aligned}
 O_i^{(1)} &= R\left([\lambda c_i^{(1)} + (1 - \lambda)c_i^{(2)}]\right) \\
 O_i^{(2)} &= R\left([\lambda c_i^{(2)} + (1 - \lambda)c_i^{(1)}]\right) \\
 R(O_i) &= \begin{cases} UB_i & \text{if } O_i > UB_i \\ LB_i & \text{if } O_i < LB_i \\ O_i & \text{else} \end{cases} \\
 \lambda &\in [0, 1] \quad (\text{uniform random}) \\
 \forall i &= [1, \dots, \dim]
 \end{aligned} \tag{4.21}$$

Geometric Crossover

Geometric crossover is similar to arithmetical crossover with exponential scaling of the parent genes as depicted in Equation 4.22 [142]. Again, this operator has been modified to include the repair technique, $R(O_i)$, described above to ensure the decision variables

remain within the problem bounds.

$$\begin{aligned}
O_i^{(1)} &= R \left([(c_i^{(1)})^\omega + (c_i^{(2)})^{1-\omega}] \right) \\
O_i^{(2)} &= R \left([(c_i^{(2)})^\omega + (c_i^{(1)})^{1-\omega}] \right) \\
R(O_i) &= \begin{cases} UB_i & \text{if } O_i > UB_i \\ LB_i & \text{if } O_i < LB_i \\ O_i & \text{else} \end{cases} \\
\omega &\in [0, 1] \quad (\text{uniform random}) \\
\forall i &= [1, \dots, \text{dim}]
\end{aligned} \tag{4.22}$$

DTGX

DTGX is a new RCGA technique that is unique to this dissertation which utilizes a truncated Gaussian so that upper and lower bounds can be enforced on each decision variable, similar to what is done using the encoding/decoding process of binary GAs. Given this, the DTGX crossover operator is able to enforce box constraints on decision variables without the use of a repair technique. A fitness-weighted average between the two parent chromosomes is then utilized to determine where the mean of the truncated Gaussian distribution will be so that the better parent is favored. This allows the DTGX operator to account for the fitness associated with each parent during the simulated reproduction process. Given that this problem will not have a fitness of zero, the formulation in Equation (4.23) can be used, however, precaution needs to be taken to avoid a division by zero in other problem formulations.

$$\bar{\mu} = \frac{\frac{P^{(1)}}{f(P^{(1)})} + \frac{P^{(2)}}{f(P^{(2)})}}{\frac{1}{f(P^{(1)})} + \frac{1}{f(P^{(2)})}} \tag{4.23}$$

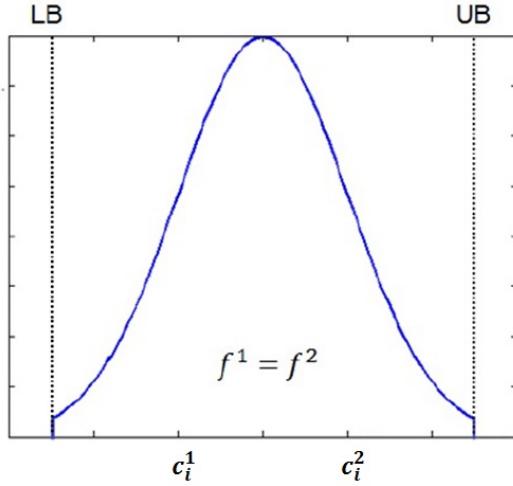


Figure 4.7. Crossover/mutation truncated Gaussian

The standard deviation σ in each dimension initially starts out large (almost uniform) to allow for an emphasis on exploration during initial generations. However, as the algorithm progresses, the standard deviations shrink to allow for increased exploitation as seen in Equation (4.24). This process was inspired by the idea of dynamic resolution [128] and uses similar logic to a simulated annealing algorithm in that it uses a set cooling schedule [179]. This idea of varying the crossover distribution with real-encoded variables is along the same lines as varying the mutation rate, given that these distributions result in any given variable having a non-zero chance of being perturbed to any value within the search bounds. The idea of adapting “mutation” distributions has been standard practice in ESSs ever since they were introduced back in the 1970s [79]. In the algorithm utilized in this chapter, the user defines a covariance “cooling” rate $\alpha = 1.001$. The standard deviation in each dimension is then reset to its initial value every time the population average fitness stagnates, or does not vary over a given number of generations by more than a set tolerance. This allows the algorithm to transition back and forth between exploration and exploitation as it samples the search space.

$$\begin{aligned}\sigma_0 &= \frac{UB - LB}{2} \\ \sigma(g) &= \sigma_0 \times \alpha^{-g}\end{aligned}\tag{4.24}$$

DGX

The DGX operator is another new technique that is identical to the DTGX operator with the exception that the tails of the Gaussian distributions are not truncated to enforce bounds on variables. Instead, the simple repair operator, $R(O)$ is utilized to set any variable in violation of an upper or lower bound equal to the bound that it has violated. This proves to be beneficial in many problems, given that it can help the algorithms explore the decision variable boundaries more effectively.

SBX

SBX has become one of the more widely used crossover operators in real coded GAs. [145], [146] and is therefore utilized in this study for comparison purposes. This operator is slightly modified from its standard form to again include the simple repair technique, $R(O)$ as depicted in Equation (4.25).

$$\begin{aligned}
 O_i^{(1)} &= R \left([(1 - \gamma)c_i^{(1)} + (1 + \gamma)c_i^{(2)}] \right) \\
 O_i^{(2)} &= R \left([(1 + \gamma)c_i^{(1)} + (1 - \gamma)c_i^{(2)}] \right) \\
 R(O_i) &= \begin{cases} UB_i & \text{if } O_i > UB_i \\ LB_i & \text{if } O_i < LB_i \\ O_i & \text{else} \end{cases} \\
 \gamma(u) &= \begin{cases} (2u)^{\frac{1}{\eta+1}} & \text{if } u \leq \frac{1}{2} \\ [2(1-u)]^{\frac{1}{\eta+1}} & \text{if } u > \frac{1}{2} \end{cases} \\
 u &\in [0, 1] \quad (\text{uniform random}) \\
 i &= [1, ..., dim]
 \end{aligned} \tag{4.25}$$

In this chapter, a dynamic aspect is added to the SBX operator so that the RCGA is able to transition from exploration to exploitation as the algorithm progresses. Beyer et al. have also used a dynamic SBXs operator that adapts itself in a way that is similar to ESs, however, it is different than the approach used in this dissertation [146]. With the SBX operator, this is done by increasing η so that the binomial distribution becomes more concentrated as the algorithm progresses. Unique to this dissertation, the RCGA in this chapter adapts the SBX

pdf as a function of generation g according to Equation (4.26) where $\alpha = 1.001$.

$$\begin{aligned}\eta_0 &= \frac{1}{UB - LB} \\ \eta(g) &= \eta_0 \times \alpha^g\end{aligned}\tag{4.26}$$

In addition to this, resets are added so that the distributions can reset and transition back to a more uniform or exploratory distribution. After the population average fitness value stagnates between generations, or changes less than a user-defined tolerance over a set number of generations, η is reset to its initial value.

4.5.3 Parallel Island Architecture

In order to leverage parallel computing to increase performance of the GAs used in this chapter, a parallel island population model is utilized. The migration frequency has been shown to have an optimal value for problems where too frequent migrations lead to the island populations becoming too similar, and too infrequent migrations result in slower learning across populations. Several previously mentioned studies have been done to look at how these parameters affect GA convergence efficiency [42], [155], [156], [158], [159]. Unfortunately, while these models and empirical results help to give insight into the relationships between number of island populations, migration rates, and population sizes, it is difficult to directly apply them to real world problems where information about building blocks is not known. These ideas are even further complicated when trying to apply them to RCGAs with new and modified operators. It is for these reasons, that a rudimentary empirical analysis is conducted on the problem of interest to determine reasonable settings for the parallel GAs used in this study.

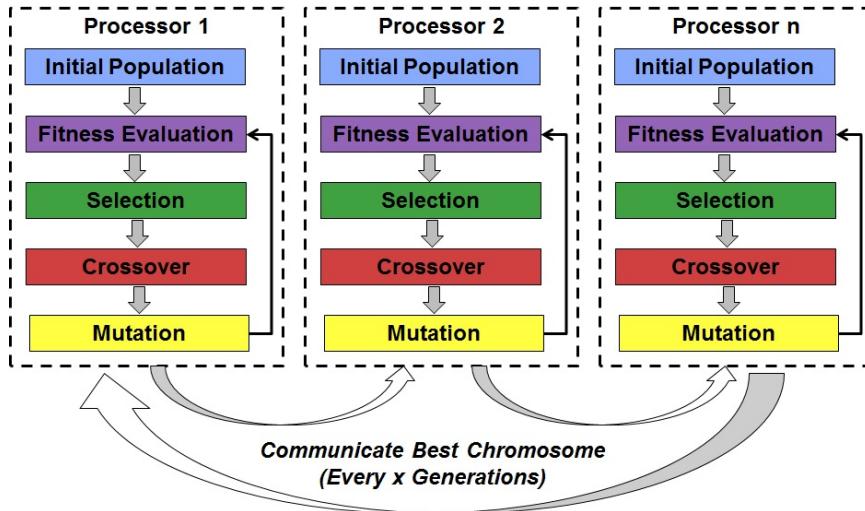


Figure 4.8. Parallel island architecture (round-robin)

All of the GAs described in this chapter utilize a simple migration algorithm in which the chromosome with the best fitness is chosen as a single migrant from each population. Using the simple round-robin mapping, this chromosome replaces a random chromosome in the neighboring island population. Cantu-Paz has shown that it is important to have fitness-based selection of the migrants, but not necessarily so for the chromosomes that they replace [158]. Not having to search for the worst chromosome in a population to replace saves computation time, and does not significantly impact convergence rates or accuracy. In addition, it has been shown that smaller migration sizes seem to perform better and that migration frequency has a larger effect on convergence results than migration sizing or number of chromosomes passed during a migration [159]. While these general observations have been determined from empirical analysis, specific trends and optimal migration frequencies are considered to be problem-dependent. For these reasons, this chapter focuses on single chromosome migrations and studies the migration frequency in accordance with what works best for the example lunar lander optimization problem under study. This chapter does not claim that this implementation is optimal in terms of the program architecture, or the computer hardware used, as even some of the most perfectly parallel problems can require a fair amount of analysis and planning to ensure this is the case [46].

4.6 Experimental GA Results

Now that the algorithms of interest have been described and the new DTGX/DGX crossover operators have been developed, it is time to begin a comparison of techniques as they apply to the example lunar lander problem. All of the above algorithms are written in C programming language. This section walks the reader through the process of tuning some of the user-defined parameters that control constraint handling and the efficiency of the parallel island implementation. While there are a number of other empirical studies that can be conducted, the focus of this chapter is on the parallel aspects of these GAs and how they affect solution quality on the lunar lander OCP. Many of the other parameters have been tuned based on results, however, this chapter does not claim that any of the above algorithms have been optimally tuned.

4.6.1 Constraint Penalty Coefficient Tuning

Empirical trials allow for appropriate settings of the adaptive penalty coefficients ($\beta_1 = 1.05$, $\beta_2 = 1.04$) to achieve a desired balance of tuning accuracy to convergence speed. The adaptive penalty method is then applied to an RCGA using a DTGX crossover operator on the lunar lander problem. Figure 4.9 illustrates how the adaptive penalty coefficient exponentially grows until solutions that are not penalized (feasible) begin to takeover the population. At this point, the penalty coefficient decreases until a good balance of penalized and unpenalized solutions have been obtained. This balancing point happens at around $p = 1$. This is to be expected given that effort has been taken with this lunar lander problem formulation to ensure that all of the states are scaled so that they are all close in magnitude. Essentially what is happening is that the algorithm is tuning the penalty parameter in a way that balances the magnitudes of the original objective function when compared to the penalty term in the penalized fitness function. After using this adaptive penalty technique to find a good penalty coefficient, it is found that by setting a fixed penalty coefficient to this value during separate runs of the algorithm actually perform slightly better than utilizing the adaptive penalty method. This is due to the fact that the penalty coefficient is instantly tuned to an appropriate value, instead of using a number of iterations to find what this value should be. Given this, the remainder of the experiments in this chapter utilize a fixed penalty term that has been set to $p = 1$. When compared to the rudimentary approach of manually tuning the penalty parameter and reevaluating the solutions, this method is much

more efficient.

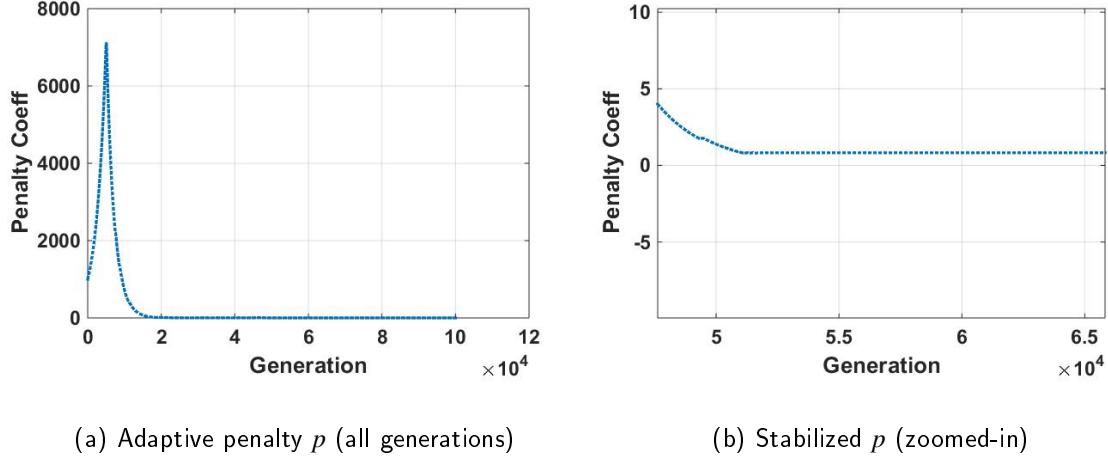


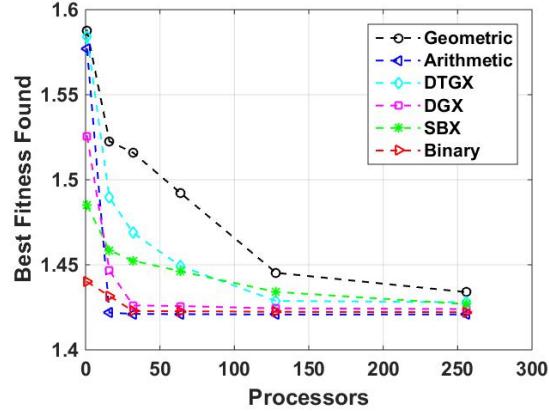
Figure 4.9. Lunar lander adaptive penalty using DTGX

4.6.2 Parallel Island Attribute Tuning

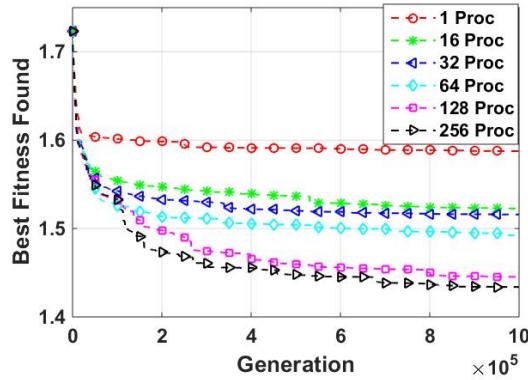
The first set of results depicted in Figure 4.10 investigates the various types of parallel island GAs and how their best fitness value for the shooting formulation of the lunar lander problem changes as the number of processors (or island populations) is increased with a fixed number of migrations (256 per 10^6 generations). Table 4.2 lists all of the specific parameters that were used to obtain these results.

	Processor Study	Migration Study	GA Type Study
Generations	10^6	10^6	10^6
Population	400	400	400
P_{cross}	0.85	0.85	0.85
P_{mut}	0.001	0.001	0.001
Pool Size	40	40	40
Migrations $\frac{1}{10^6}$ Gen.	256	Ref. Fig. 4.11	Ref. Fig. 4.12
Processors	Ref. Fig. 4.10	256	256
Penalty Parameters	$10^0, \beta_1 = \beta_2 = 1.0$	$10^0, \beta_1 = \beta_2 = 1.0$	$10^0, \beta_1 = \beta_2 = 1.0$
Time Steps (N)	50	50	50

Table 4.2. Method comparison GA attributes



(a) Effects of processor number on various GA types



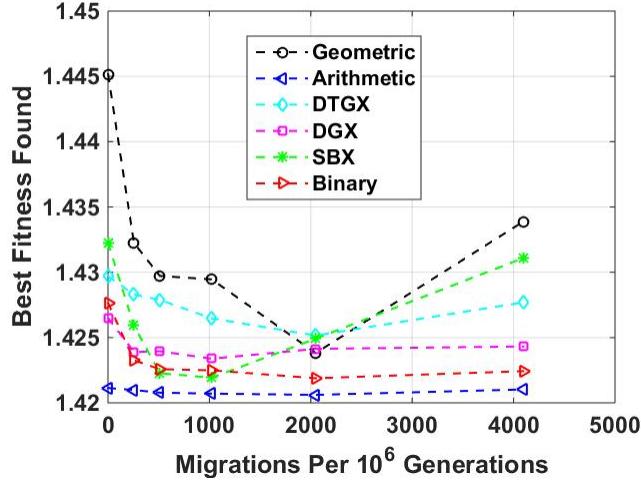
(b) Geometric RCGA fitness evolution

Figure 4.10. Effects of number of processors on GA performance over 10^6 generations

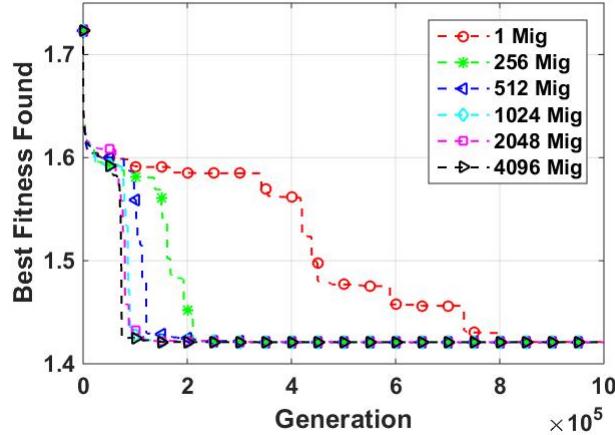
Figure 4.10a illustrates the best fitness found for each GA variant over 10^6 generations after averaging the results for 10 consecutive runs. It can be seen that the performance tends to improve as more and more island populations (processors) are added. In fact, some of these GA types obtain significant gains in performance and accuracy through the utilization of additional processors. This indicates that the use of parallel computation can increase GA robustness and reduce the sensitivity of GA parameter tuning. The algorithm types that perform the worst when run in serial, appear to benefit the most from the incorporation of parallel processing. A more detailed plot of how the best fitness value evolves over the 10^6

generations for the Geometric RCGA is illustrated in Figure 4.10b. It appears that more optimally tuned algorithms such as the simple binary GA have less to gain from massive parallelization than do algorithms that may be less optimally tuned. The binary GA is likely to be more optimally tuned, given that many of the parameter values chosen for this study were based on literature that was specific to binary GAs. However, it can also be determined that the return on investment regarding the utilization of more and more processors seems to diminish. With a limited amount of computing resources, a practical number of processors to use for the remainder of this study is 256.

The next set of results is used to generate an empirical understanding of how many migrations should be used for a given number of island populations (256) over a given number of generations (10^6). Table 4.2 describes the various GA parameters that were used to obtain the results depicted in Figure 4.11.



(a) Effects of migration frequency on various GA types



(b) Arithmetic RCGA fitness evolution

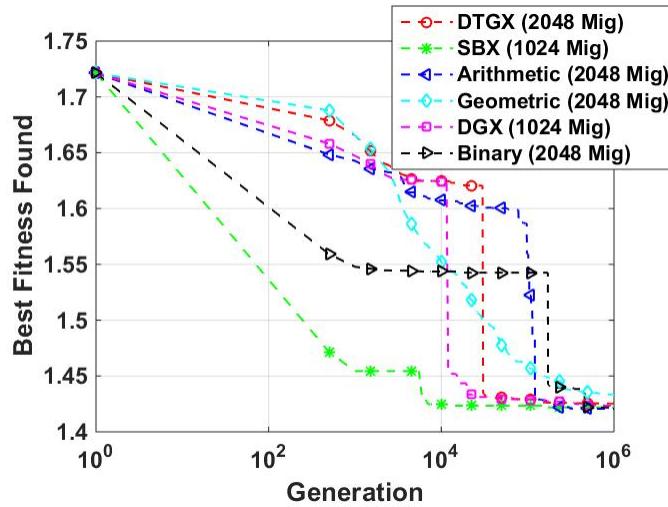
Figure 4.11. Effects of number of migrations (per 10^6 generations) on GA performance

It can be seen from Figure 4.11a that there is also a diminishing return as the migration frequency is increased beyond a certain point. In fact, in many of the algorithms, the performance begins to deteriorate as the frequency of migrations increases beyond some ideal setting. The reason for this is that the populations become similar very quickly with migration frequencies that are too high, and the benefit of population-to-population

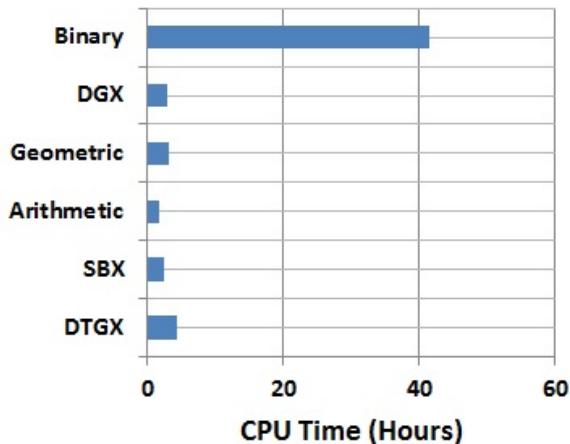
diversity is lost. On the other end of the spectrum, too few migrations can slow convergence by making the parallel island GA behave like multiple, independent GAs that do not share information until they have completed all of their generations. For this reason, there is an ideal migration frequency for each problem and algorithm type. Again, the combination of parallel GA attributes that optimizes the performance of each GA type is difficult to predict for a given problem based on theoretical analysis alone. It is for this reason that the above trades are conducted so that a fairly efficient parallel GA can be developed for each type of algorithm. While Figure 4.11a indicates that the Arithmetic RCGA does not appear to be significantly affected by the number of migrations per run, the fitness evolution in Figure 4.11b provides a more detailed illustration. It can be seen that convergence speeds are improved with an appropriate number of migrations, despite the fact that they all eventually converge to similar values after enough generations when using 256 processors. This improvement would be more noticeable if the user decided to set the stopping criteria to a lower number of generations e.g. 200,000 in Figure 4.11b.

4.6.3 GA Type Comparison

The results and exercises above allow for tuning the appropriate numbers of processors and migrations used for each the parallel GA architectures. The next set of results illustrated in Figure 4.12 is used to compare the effectiveness of the various parallel genetic algorithms described in this chapter when applied to the shooting method formulation of the lunar lander problem. The parameter settings for these various algorithms can be found in Table 4.2.



(a) Best fitness vs generations



(b) CPU time required for 10^6 generations

Figure 4.12. Comparison of GA types for lunar lander problem

Figure 4.12 illustrates how the various methods compare when solving the lunar lander problem using the shooting method averaged over 10 trials. It must be noted that these algorithms have not been optimally tuned with respect to all parameters. With this, less emphasis should be put on which specific GA type outperforms the others. More important for this study is to determine how well these algorithms perform when coupled with parallel

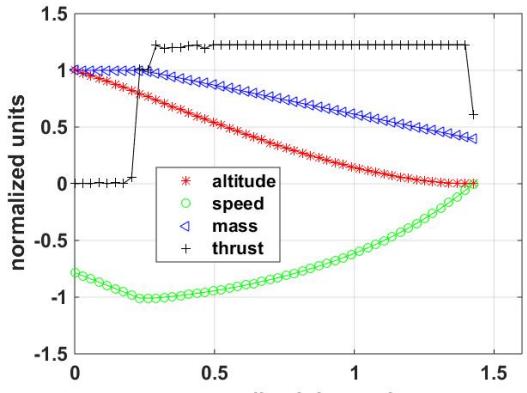
processing, and whether or not parallelization is enough to overcome less than optimal GA parameter settings. It can be seen that the RCGA with arithmetic crossover seems to achieve the best results both in terms of fitness found and computational time. Its computationally simple operators of multiplication and addition run faster than the distribution-based and geometric operators that involve exponentials. The Arithmetic RCGA is a prime example of how parallel computing can benefit these GAs and overcome many of their weaknesses. Referring back to Figure 4.10a, the Arithmetic RCGA was one of the worst performing GAs when run in serial on a single processor. SBX achieves the fastest initial convergence, meaning that it could be considered the best method if the algorithms are run for a smaller number of generations. This type of crossover has been shown to be very efficient when compared to other RCGA crossover types and has become fairly popular [171]. The binary GA also performs well in terms of best fitness found, but takes significantly longer to run for a couple of reasons. First of all, a decoding process must be performed every time the fitness function is evaluated. In addition, each decision variable is encoded using 32-bit precision making some of the operators (e.g. mutation) take longer given that they have to step through larger numbers of variables. Table 4.3 further summarizes these results by showing the best single case of the 10 trials for each GA type where F_{Best} is the best fitness found by the algorithm and F_{Best} error is the difference between F_{Best} and the known optimal as a percentage.

Method	Migrations 10^6 Gen.	F_{Best}	F_{Best} Error (%)	Avg. CPU Time (hours)	Time Steps
DGX	1024	1.4234	0.2183	2.9123	50
DTGX	2048	1.4252	0.3450	4.3799	50
SBX	1024	1.4219	0.1127	2.3756	50
Arithmetic	2048	1.4206	0.0211	1.7689	50
Geometric	2048	1.4245	0.2957	3.3174	50
Binary	2048	1.4218	0.1056	41.6058	50

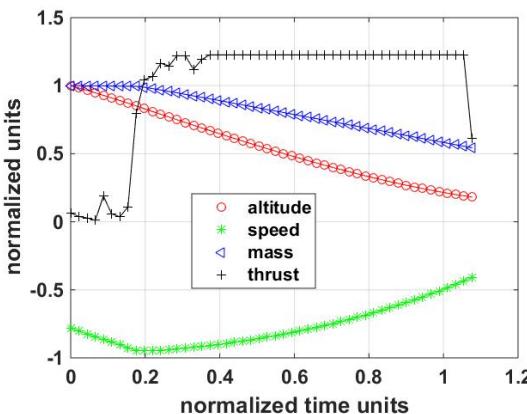
Table 4.3. GA type comparison results

Figure 4.13 compares the best parallel, Arithmetic RCGA lunar lander problem solution with the best solution found using a serial Arithmetic RCGA. It can be seen that the parallel implementation (Figure 4.13a) significantly improves the optimal trajectory without any adjustment of the original GA parameters such as: population size, tournament pool size, crossover rate, mutation rate, etc. The serial Arithmetic RCGA (Figure 4.13b) fails to find

a feasible solution after 10^6 iterations. This further indicates the usefulness of parallel processing when applied to these types of evolutionary algorithms where *a priori* optimal parameter tuning can be difficult.



(a) 256 Processors (feasible)



(b) 1 Processor (infeasible)

Figure 4.13. Arithmetic RCGA 50-node lunar lander trajectory: parallel(a) vs. serial(b)

4.7 Conclusion

This chapter illustrates how a continuous optimal control problem can be transformed into an unconstrained NLP using direct shooting and exact penalty methods. It also introduces and investigates various types of exact and inexact penalty functions in terms of how they

perform on highly constrained test problems from literature using parallel GAs. From here, two new real variable crossover techniques are introduced (DGX and DTGX) are further developed to address the issues of transitioning between exploration and exploitation of the problem search space. This study uses a simple parallel island architecture to investigate the effectiveness of six different parallel GA types when applied to an example lunar lander problem. The results indicate that parallel computing improves the robustness, convergence times, and accuracy of all GA types considered in this chapter. It alleviates the requirement for the practitioner to perfectly tune all of the user-defined parameters in order to obtain near-optimal results. Furthermore, the implementation of parallel computing had the largest impact on the GAs that had the worst performance when run in serial. The findings of this study are encouraging in that they show the usefulness of parallel GAs and their effectiveness in solving optimal control problems. From this point, the dissertation research trajectory will shift toward the analysis and development of a set of algorithms that are closely-related to GAs known as ESs. The DGX/DTGX operators introduced in this chapter will become increasingly important throughout this dissertation, as they lead to innovative unscented sampling ideas within the constructs of both ESs and RCGAs developed and applied in Chapter 6 and Appendix C respectively.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 5: Evolution Strategies

5.1 Introduction

Around the time that John Holland introduced the GA in the United States, a similar type of evolutionary algorithm was being developed across the Atlantic in Germany [39], [180]. This algorithm was initially formulated to work with continuous “real-coded” variables and was formally introduced by Rechenberg in 1973 as “Evolutionsstrategie,” or ES [40], [79]. Rechenberg originally began developing the rudimentary version of this algorithm in 1964 to optimize a 3-D body in a wind tunnel with regard to drag per unit volume. His original algorithm worked with only two simple rules: randomly perturb the design variables with primarily small variations, and only replace the current design solution with one that outperforms the current one. [180] This idea became the underpinnings of the original ES introduced by Rechenberg and was further developed by Schwefel in 1974 to incorporate multiple parents and offspring [79]. ESs are similar to GAs in that they both attempt to simulate biological evolution processes that have been observed in nature. The classical ES relies on Gaussian sampling of the problem search space using continuous variables, whereas GAs were initially applied to combinatorial type problems with binary encoded variables.

The use of Monte Carlo concepts to sample the domain of objective functions is standard practice in ES algorithms. This approach uses the idea that a Gaussian distribution allows the algorithms to closely parallel the workings of genetic mutation and evolution in nature e.g. large genetic changes as a result of a mutation are far less likely than small changes [151]. The technique of random sampling from a normal distribution also maintains a fair balance between exploration and exploitation, enabling the ES to escape locally optimal points if a large enough mutation occurs.

This chapter begins with an introduction to Rechenberg’s original ES and then transitions to the multiple parent/offspring algorithms that have been introduced over the years. After this, it introduces the various ES operators including: mutation, selection, and recombination.

From here, the chapter describes the primary covariance adaptation techniques as they were developed in chronological order so a logical progression can be achieved. The chapter then details the current state of the art in ES algorithms by looking at the CMA-ES, NES, and exponential natural evolution strategy (xNES). The chapter concludes by presenting a top-level overview of ES theory and dynamics modeling techniques.

5.2 Conventional Evolution Strategy

An ES shares the overarching fundamentals of other evolutionary algorithms in that it starts with an initial candidate solution or set of candidate solution vectors and evolves them toward solutions that achieve better objective function values. Equation (5.1) illustrates the standard form of an unconstrained optimization problem. With \bar{x} being a potential solution vector, the objective is to find the global optimum \bar{x}^* such that $f(\bar{x}^*) \leq f(\bar{x})$ for all possible values of \bar{x} in the admissible set. It must be noted that most optimization problems are subject to any number of inequality and equality constraints. However, for the initial description of these algorithms, the optimization problem is treated as an unconstrained one. Given this, Appendix B summarizes a number of constraint handling techniques for evolutionary algorithms.

$$\begin{aligned} & \text{Min } f(\bar{x}) \\ & s.t. \\ & \bar{x} \in \mathbb{R}^n \end{aligned} \tag{5.1}$$

The standard ES algorithm can be summarized by four steps as depicted in Figure 5.1. The initialization step represents starting with an initial set of candidate solution vectors or “parent population” and initial parameters that define how the other steps of the ES will behave. Without any prior knowledge of the optimization problem at hand, the standard practice is to randomly select the initial set of parent vectors, or candidate solution vectors from the search space using a uniform random distribution [79]. It must be noted that a uniform distribution requires bounded variables. These bounds are typically selected by the practitioner so that they define a large region where the optimal solution may reside. After an initial set of candidate solution vectors have been determined and the other ES parameters

have been set, the next step in the algorithm, fitness evaluation, is used to determine the objective function value for each trial solution vector. After the objective function values or “fitness” has been determined for the population, the selection step emulates the survival of the fittest aspect that manifests itself in biological evolution. This is done by refining the set of solution vectors in a way that favors the survival of candidate solution vectors that have the best objective function values or fitness values. Typically for this step, ESs utilize a sorting algorithm that orders the population on the basis of objective function value. From this point, a higher-performing subset of the candidate solution vectors are selected as “parent” solution vectors. [79] The next phase of the algorithm, offspring generation, consists of perturbing or “mutating” the parent solution vectors in a way that creates a new set of candidate solution vectors or “offspring”. This represents a new population for the next iteration of the algorithm. In an ES, the offspring are often generated by sampling from a probability distribution. The vast majority of ESs utilize a multidimensional normal distribution with the mean value equal to the parent solution vector(s). The covariance determines the distance of the offspring from the parent and is initially set by the user. The covariance is then allowed to evolve using any number of various covariance evolution schemes [181]–[184]. The algorithm then goes back to the fitness evaluation step and continues this iterative process until some stopping criteria is met. The various stopping criteria for ESs are the same as the ones described in Section 3.8 for GAs.

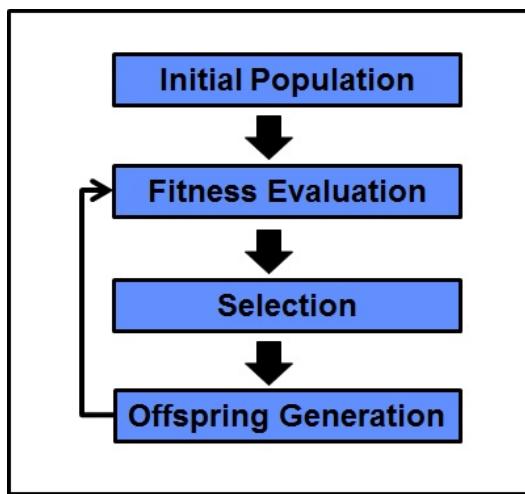


Figure 5.1. Standard ES algorithm

5.3 ES Naming Convention

There is a standard notation for ESs that helps the user identify how the ES is setup. The generic notation is $(\mu; \lambda)$ where μ signifies the number of parent solution vectors, λ is the number of offspring solution vectors. In this notation a “+” means that the parents will compete against the offspring during the selection process, whereas “,” indicates that the selection process only considers the offspring. More recently, there is also a $(\mu/\rho; \lambda)$ ES. This algorithm selects a subset, ρ , from the λ offspring that have been created. The algorithm then applies a recombination technique to this subset of selected offspring to create a new parent solution vector that is representative of each subset of chosen offspring. This process of selecting a subset and recombining them into a parent vector is repeated μ times to create μ new parent vectors. These various forms of ESs will be discussed in more detail throughout the following sections. As an example, a $(1 + 1)$ ES indicates that there is one parent, $\mu = 1$, one offspring, $\lambda = 1$, and it uses the + selection scheme, meaning that both the offspring and parent vector are considered for selection. In this chapter, a generic integer value for the number of parents or offspring that is greater than one is indicated by leaving the symbol in the naming convention. For example, a $(1, \lambda)$ ES indicates a group of ESs where $\mu = 1$, $\lambda > 1$, and only the offspring are considered for selection.

5.4 $(1 + 1)$ ES

There are a number of variations to the simple ES. They are, however, all built upon the idea of the original $(1 + 1)$ ES further depicted in Figure 5.2. In the case of the $(1 + 1)$ ES, an initial parent solution vector is created, typically by using a uniform random distribution. In this case, the parent vector consists of a candidate solution vector and an associated, single covariance parameter for all of the decision variables, $[\bar{x}_{parent}, \sigma]$. The first version of the $(1 + 1)$ ES used a spherical distribution with a single σ value, where the diagonal elements of the full covariance matrix, C_{parent} were all equal to σ^2 . The typical Gaussian mutation process for a $(1 + 1)$ ES uses an n -dimensional normal distribution $\mathcal{N}(\bar{x}_{parent}, C_{parent})$ that is centered on the parent solution vector \bar{x}_{parent} and has a covariance matrix, C_{parent} , which is associated with that particular parent vector. A new solution vector $\bar{x}_{offspring}$ is created from a random sample of this normal distribution centered on the parent. As illustrated in Figure 5.2, ESs often adapt the covariances that are associated with each solution vector along with the adaptation of solution vectors. This process of adapting the covariance can

be done using a number of different methods that are described in Section 5.11. More specifically, the process known as the one-fifth success rule [40] was originally utilized to adapt the covariance parameter in the original $(1 + 1)$ ES and will be addressed in Section 5.11. The solution vector that produces the best objective function value of the two (one parent + one offspring) is then selected to be the parent that is used to produce the offspring for the next “generation.” At this point, there is now a new population (one parent vector and one offspring vector) that is ready to go through the entire process again. Each iteration of this loop is referred to as a “generation”. This population is evolved in this manner until some type of exit criteria is reached (as discussed in Section 3.8), whether it be a maximum number of generations, or a stagnation in terms of the population fitness from generation to generation.

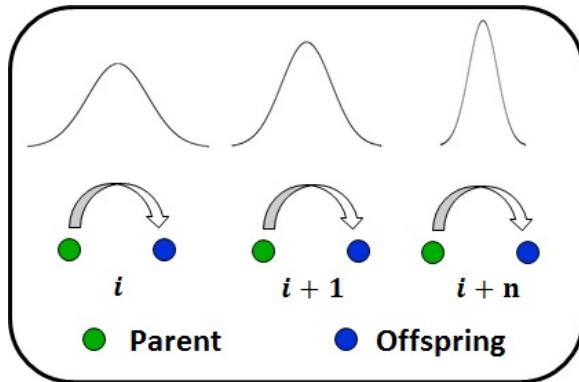


Figure 5.2. $(1+1)$ ES illustration

5.5 $(\mu + 1)$ ES

Knowing the basic framework of a $(1 + 1)$ ES, it is now easier to imagine that there are a number of variations to this idea. The earliest ES with more than two solution vectors present in a population at a given time was the $(\mu + 1)$ ES introduced by Rechenberg [40]. This algorithm randomly selects two of the μ parents to undergo recombination and produce a single offspring. Common recombination techniques will be introduced and described in Section 5.10. From this point, a simple selection operator eliminates the worst individual from the population to maintain a constant population size. This process then repeats itself by selecting two more parents to create a new offspring vector.

5.6 (μ^+, λ) ES

Instead of taking a single sample from the normal distribution to create a single offspring, why not take multiple samples? This is the idea behind the $(1, \lambda)$ ES or $(1 + \lambda)$ ES, where these ESs produce multiple (λ) offspring [79]. Another variant that more closely parallels a real-coded GA is the (μ, λ) ES or $(\mu + \lambda)$ ES [79]. The simplest form of these algorithms is illustrated in Figure 5.4 where $\mu < \lambda$ and the population is improved over time through a selection operation. However, there are a number of variations to this simple idea where different methods can be used to produce the next generation of parent vectors from the offspring vectors.

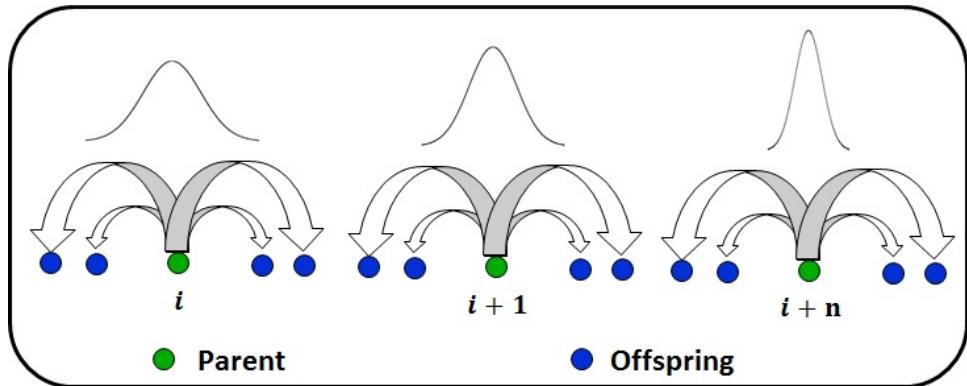


Figure 5.3. Single parent/multiple offspring ES illustration

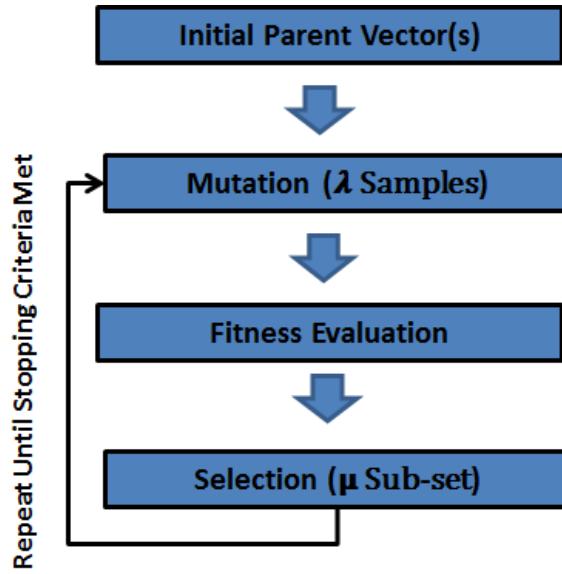


Figure 5.4. Model of a $(\mu + \lambda)$ ES

5.7 $(\mu/\rho, \lambda)$ ES

With the aspect of having larger numbers of offspring than parent vectors comes the need to reduce the number of offspring through selection and the potential to create new parent vectors through various recombination techniques. With these types of algorithms, the simple delineation between parent vectors and offspring becomes a little more complicated given that there is now an additional step. In order to help clarify this distinction a new symbol is introduced. With this terminology, μ represents the number of parent vectors that undergo mutation, λ is the total number of offspring vectors generated from mutation of all parents, and ρ is the number (or subset) of selected offspring vectors used to generate μ new parent vectors through a process called recombination. Many of the more recently developed ESs utilize variations of this basic ES construct [50], [183]. Figure 5.5 illustrates a more specific ES algorithm that begins with an initialization step. It then enters into the iterative phase where it produces sample points through a mutation operator. Next, the $(\mu/\rho, \lambda)$ ES produces the next parent vector (or set of μ parent vectors) through selection and recombination. It must be noted that in Figure 5.5, the selection and recombination operators will repeat μ times, one iteration to create each new parent vector. If the algorithm

uses more than one parent vector, the selection scheme that is used to select the subset of ρ offspring should have some element randomness introduced to avoid selecting the same ρ offspring for the creation of all of the parents which would result in identical parents. The other option would be to use a recombination operator that introduces some randomness, and could therefore produce different parent vectors from the same set of ρ chosen offspring vectors (e.g. dominant discrete recombination [180]). The selection and recombination operators will be discussed in more detail later in this chapter.

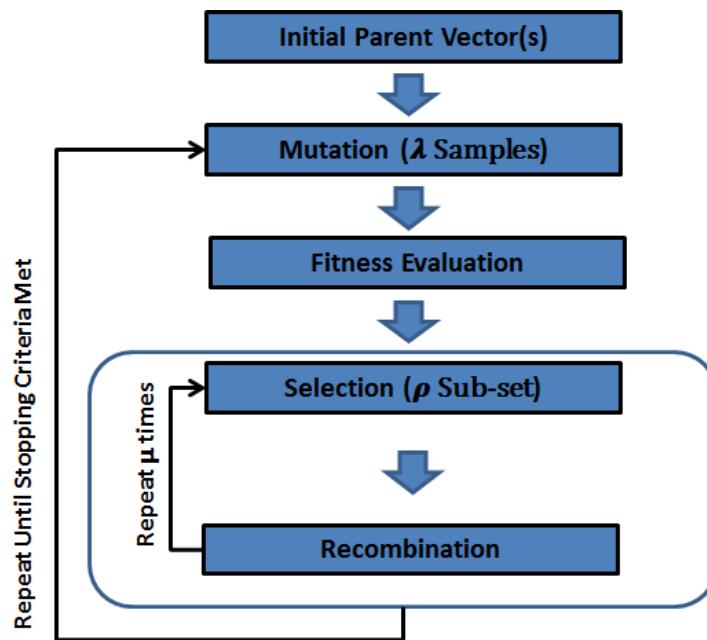


Figure 5.5. Model of a $(\mu/\rho + \lambda)$ ES

5.8 Mutation

The mutation operator is a key part of ES algorithms and is the primary source for generating the next set of offspring. In Beyer's book on ES theory [151], he develops several properties that good mutation operators should have, but also states that: "Not all of these requirements are absolutely important, nor are they definitely necessary." [151] The following list is paraphrased from Beyer's book [151] as follows:

Reachability The current state of \bar{x}_{parent} should be able to transition to any other state

within the search domain within a finite number of generations $g < \infty$.

Scalability The step size (e.g. covariance for normal distributions) should be a tunable parameter that has the ability to be tuned toward a locally optimal mutation strength setting.

Unbiased The mutation operator should be maximally unbiased and be the one with the maximum entropy, or ability to best represent the current state of knowledge. Beyer uses the word entropy in reference to the maximum entropy principle, where a Gaussian distribution has been shown to have the maximum entropy for continuous variables [151]. This is one of the reasons why Gaussian distributions are often used in ESs operating in continuous search spaces.

Symmetry Coupled with unbiased distributions, this property indicates that the expectation of state changes should be zero, or symmetric about the parent vector. It must be noted that several modern ESs do not follow this guidance (e.g. cumulative self-adaptation evolution strategy (CSA-ES) [181], CMA-ES [183], and NES [50]).

For the above reasons, random sampling from normal distributions is a popular choice for mutation operators internal to ESs working in continuous search domains [49], [151], [180]. The decision variables of each parent chromosome, \bar{x}_{parent} , typically define where the mean of the mutation distribution will lie in the n-dimensional search domain. One primary difference between ESs and RCGAs is that each chromosome or parent vector in a typical ES also carries with it “endogenous strategy parameters” that define the shape of the mutation distribution, or mutation strength [180]. They are referred to as endogenous parameters given that they can be changed or adapted internal to the algorithm, in support of Beyer’s scalability requirement. Whereas the “exogenous” strategy parameters are things that are defined by the user, external to the algorithm such as the number of parents, μ , or offspring, λ , and are unchanged during the execution of the algorithm [180]. While various mutation distributions may be used, a standard Gaussian distribution is assumed for this chapter and the endogenous strategy parameters associated with it are ones that describe the distribution mean, \bar{x}_{parent} , and covariance matrix C . It will be seen that the covariance matrix defines how a Gaussian mutation operator will explore the search domain of interest, and will be used to guide the search toward favorable regions of the search space in more modern ESs. In simplified cases, strategy parameters describing covariance can be a single scale value, σ , for all dimensions, or they can be a full covariance matrix used to create

various mutation distributions [180]. The endogenous strategy parameters that describe the mutation distributions can be adapted throughout the algorithm execution using a number of techniques. Several of these covariance adaptation techniques are described in Sections 5.11 through 5.13.

One primary characteristic that distinguishes the ES mutation operator from that of GAs is that the mutation operator is typically the operator used to spawn the next set of offspring by sampling the search space and is the primary means of search for an ES. Some authors refer to this as “asexual” reproduction given that new trial solution vectors are spawned from a single parent chromosome through this mutation operator [185]. Whereas the mutation operator within a GA is primarily utilized to escape local optima by slightly perturbing a small number of chromosomes from generation to generation with low probability.

5.8.1 Spherical Mutation

In the most basic sense, the perturbed parameters in each search dimension obey the characteristics of the standard Gaussian probability density function $f_{\mathcal{N}(0,1)}(z_i)$ outlined in Equation (5.2) where $\mathcal{N}(0, 1)$ indicates a random selection from a normal distribution with a zero mean and a standard deviation of one.

$$f_{\mathcal{N}(0,1)}(z_i) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2} (z_i)^2\right) \quad (5.2)$$

Equation (5.3) illustrates the process of creating new offspring solution vectors using a simple spherical mutation distribution. The notation $\bar{z}^{(k)} \in \sigma(\mathcal{N}_1(0, 1), \dots, \mathcal{N}_n(0, 1))$ indicates that each component of z_i is randomly selected from a normal distribution $\mathcal{N}_i(0, 1)$ and then scaled by a single standard deviation, σ . From this point, the random vector, $\bar{z}^{(k)}$, is shifted by the distribution mean or parent vector, \bar{x}_{parent} , to create the new offspring solution vector, $\bar{x}_{offspring}^{(k)}$. It must be noted that a single parent case is assumed to illustrate these mutation techniques. In this simplest case, each decision variable has the same standard deviation, σ , creating a perfectly symmetrical, n-dimensional, normal distribution around the parent vector, where n is the solution vector dimension.

$$\begin{aligned}
\bar{z}^{(k)} &\in \sigma(\mathcal{N}_1(0, 1), \dots, \mathcal{N}_n(0, 1)) \\
\bar{x}_{offspring}^{(k)} &= \bar{x}_{parent} + \bar{z}^{(k)} \\
k &= (1, \dots, \lambda)
\end{aligned} \tag{5.3}$$

The spherical mutation technique makes it so that the parent vector only carries one endogenous strategy parameter that is a single standard deviation for all the dimensions. This results in spherical, normal mutation distributions. Figure 5.6 illustrates a 2-D version of this idea, where the shaded area indicates a given probability interval used to illustrate the distribution shape associated with this concept. In actuality, these Gaussian distributions span an infinite space.

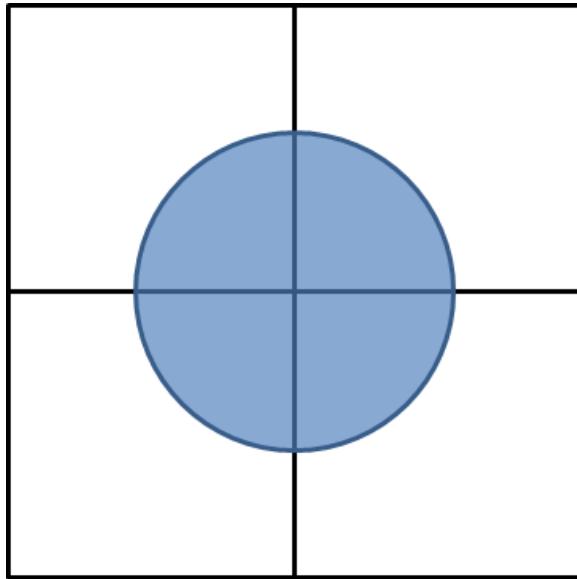


Figure 5.6. Spherical mutation illustration

5.8.2 Axis-Parallel Mutation

Next, a level of complexity can be added by allowing each decision variable, or component of the random vector, $\bar{z}^{(k)}$, to have its own associated standard deviation so that there are now n strategy parameters per parent vector. This allows for n-dimensional ellipsoidal distributions to be utilized to allow some decision variables to vary more than others as

depicted in Equation (5.4).

$$\begin{aligned}
 \bar{z}^{(k)} &= (\sigma_1 \mathcal{N}_1(0, 1), \dots, \sigma_n \mathcal{N}_n(0, 1)) \\
 \bar{x}_{offspring}^{(k)} &= \bar{x}_{parent} + \bar{z}^{(k)} \\
 k &= (1, \dots, \lambda)
 \end{aligned} \tag{5.4}$$

Figure 5.7 illustrates a simple 2-D example where each axis of the distribution is allowed to carry a different standard deviation value, creating an axis-parallel ellipsoid. Again, the shaded area of this figure represents a confidence interval, and does not illustrate the entire probability distribution.

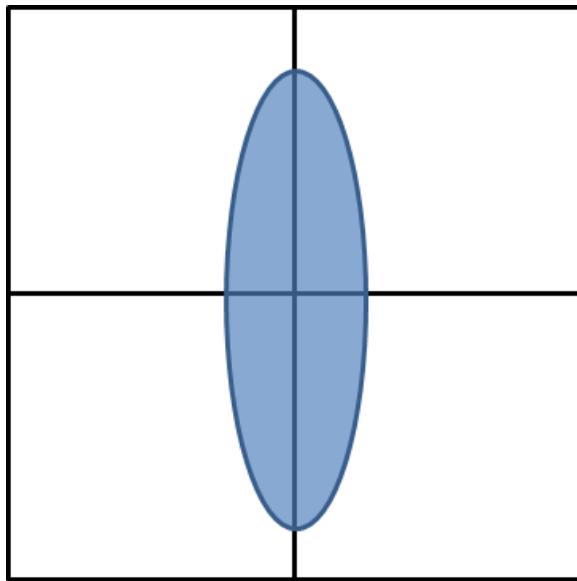


Figure 5.7. Axis-parallel mutation illustration

5.8.3 Full Covariance Mutation

Lastly, one further level of complexity can be added to the Gaussian mutation scheme so that rotations or correlations between decision variables can be accounted for as well. Such a mutation technique allows for increased flexibility in an ES's ability to guide the search based on observed trends in the fitness landscape. The more recent ESs (CMA-ES

and NES/xNES) tend to utilize this approach of using a full covariance matrix so that the algorithms have the ability to fully adapt the mutation distributions based on the fitness landscapes [50], [183].

In order to achieve this, the mutation technique utilizes the full $n \times n$ covariance matrix, C , as an endogenous strategy parameter. In addition, the full covariance mutation technique uses a multidimensional normal distribution to create n -dimensional random vectors that are then used to generate new offspring vectors from a parent vector. Equation (5.5) illustrates the standard, multidimensional normal probability density function typically used in the full covariance mutation scheme. It must be noted that the standard normal pdf represents the case with a zero mean and a covariance equal to an $n \times n$ identity matrix, I .

$$f_{\mathcal{N}(0,I)}(\bar{z}) = \frac{1}{\sqrt{2\pi^n}} \exp\left(-\frac{1}{2}\bar{z}^T \bar{z}\right) \quad (5.5)$$

For a full covariance mutation distribution, the random vector, $\bar{z}^{(k)}$, is randomly selected from the n -dimensional standard normal distribution outlined in Equation (5.5). As illustrated in Equation (5.6), the offspring vectors are then produced through an affine transformation where the random vector $\bar{z}^{(k)}$ is scaled by the square root of a given covariance matrix, C , and shifted by a given mean, \bar{x}_{parent} .

$$\begin{aligned} \bar{z}^{(k)} &= \mathcal{N}(0, I) \\ \bar{x}_{offspring}^{(k)} &= \bar{x}_{parent} + \sqrt{C} \bar{z}^{(k)} \\ k &= (1, \dots, \lambda) \end{aligned} \quad (5.6)$$

Figure 5.8 illustrates a simple 2-D example where a full covariance matrix is utilized to allow for correlated distributions, thus maximizing the ability to fully shape and control the mutation distribution shape. Again, the shaded area of this figure represents a confidence interval, not the entire probability distribution.

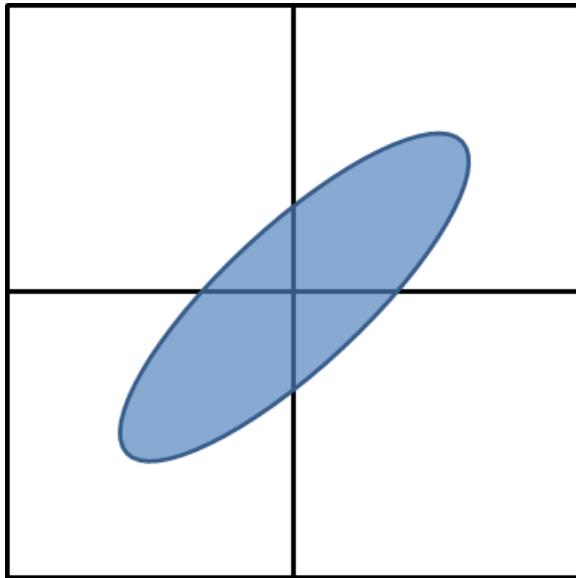


Figure 5.8. Full covariance mutation illustration

In this model \sqrt{C} represents a factorization of the covariance matrix that can be handled a couple of different ways. One common method is to utilize Cholesky factorization where A is the Cholesky factor as described in Equation (5.7). [186] This method works well, but requires the updated C matrix to be positive definite and can have some computational precision robustness issues. These issues can arise when the ESs update the covariance due to machine precision in numerical methods where terms of the matrix might become slightly negative (e.g. -10^{-14}) and should be considered zero for practical purposes. Given this, care should be taken to properly condition/scale matrices to mitigate these issues.

$$\begin{aligned}
 C &= AA^T \\
 \bar{x}_{offspring}^{(k)} &= \bar{x}_{parent} + A\bar{z}^{(k)} \\
 k &= (1, \dots, \lambda)
 \end{aligned} \tag{5.7}$$

Eigenvalue/Eigenvector decomposition can also be used to avoid some of the computational issues that may arise when using Cholesky factorization, but still requires that the C matrix is positive definite. This technique is described in Equation (5.8) [187]. In this method G

is a matrix whose columns are the orthonormal eigenvectors of C and D^2 is a matrix with the corresponding eigenvalues of C along the matrix diagonal.

$$\begin{aligned} C &= GD^2G^T \\ C^{\frac{1}{2}} &= GDG^T \\ \bar{x}_{offspring}^{(k)} &= \bar{x}_{parent} + GDG^T \bar{z}^{(k)} \\ k &= (1, \dots, \lambda) \end{aligned} \tag{5.8}$$

5.9 Selection

Beyer and Schwefel point to truncation selection as being the selection method that is typically utilized in conventional ESs [180]. This selection technique implements a survival of the fittest approach by sorting the population to be considered and then only selecting the top ρ individuals that will influence the next set of offspring, either directly or after going through a recombination operator. The population that is considered for selection depends on whether the ES is a $(\mu/\rho, \lambda)$ type or a $(\mu/\rho + \lambda)$ type. As mentioned previously, the ‘,’ ES indicates that only the λ offspring will be considered for selection and the parents expire after a single generation. However, with the ‘+’ ES, both parents and offspring are considered for selection which in turn is a natural way of enforcing elitism given that truncation selection always selects the best individuals of the population as a result of fitness-based sorting. As can be imagined, ESs can also implement other selection operators such as tournament and proportional selection that introduce an element of randomness into the selection process. However, this is not as common as truncation selection for ESs [36].

5.10 Recombination Techniques

Recombination techniques share some similarities with crossover operators in GAs but also have several distinct differences. For example, it is common practice in modern ESs to produce a single parent vector from a higher-performing subset of multiple offspring, ρ , whereas GAs typically produce two offspring from two parents [180]. Second, the distribution-based crossover techniques that have been discussed in Chapters 3 and 4 for

RCGAs are more closely linked to the mutation step in ESs. In an empirical analysis, trying to answer questions regarding the usefulness of multi-parent recombination within the construct of an ES, it was found on the majority of problems tested that using $\rho > 1$ parents did improve algorithm performance [188]. This has also been shown theoretically for specific ES types and fitness function classes [151]. In addition, the empirical study [188] found that the benefit of increasing the number of ρ selected offspring beyond two is highly dependent on the specific combination of recombination operator and fitness function. Eiben et al. identify three broad groups of recombination operators that allow for a tunable number of selected offspring where the term “ ρ -ary” is in reference to the number of solution vectors being recombined into a new parent vector. The three groups they identified are: operators based on occurrences of a particular allele (voting recombination, ρ -ary discrete recombination, and scanning crossover), operators based on segmenting and recombining parents (standard crossover, diagonal crossover, and (r, s) -segmentation crossover), and operators based on numerical operations such as averaging (recombination by averaging, ρ -ary intermediate recombination, and ρ -ary global intermediate recombination) [185].

To make a long story short, there are a large number of potential recombination operators that have been used, or could be applicable to various types of ESs. However, many of these recombination operators are not frequently used in standard ES algorithms and do not warrant detailed descriptions in this chapter. Beyer identifies the two primary types of recombination utilized in ESs as dominant (discrete) recombination and intermediate recombination [151]. However, the literature survey done on this topic has produced several variations to the standard intermediate recombination operators that have been applied in more recent forms of ES algorithms. For this reason, they are described as separate techniques for the purposes of distinction in this overview chapter. While not typically used in ESs, a number of the crossover operators utilized in GAs can, in theory, be applied to these algorithms as well [49]. Given that this is not common practice, and many of these operators have been previously discussed in Chapter 3, this section only outlines the common recombination operators that are typically used in ESs.

5.10.1 Dominant (Discrete) Recombination

Dominant recombination, or sometimes referred to as discrete recombination given that the individual decision variables are randomly selected from the parent pool and pieced together

as discrete chunks to produce the offspring vector [180]. Figure 5.9 illustrates this process where each given color represents a parent vector. As can be seen, this recombination operator shuffles the order of decision variables, but is heavily dependent on mutation to introduce new decision variable values into the population from one generation to the next. This operator is able to work with both continuous and discrete decision variables.

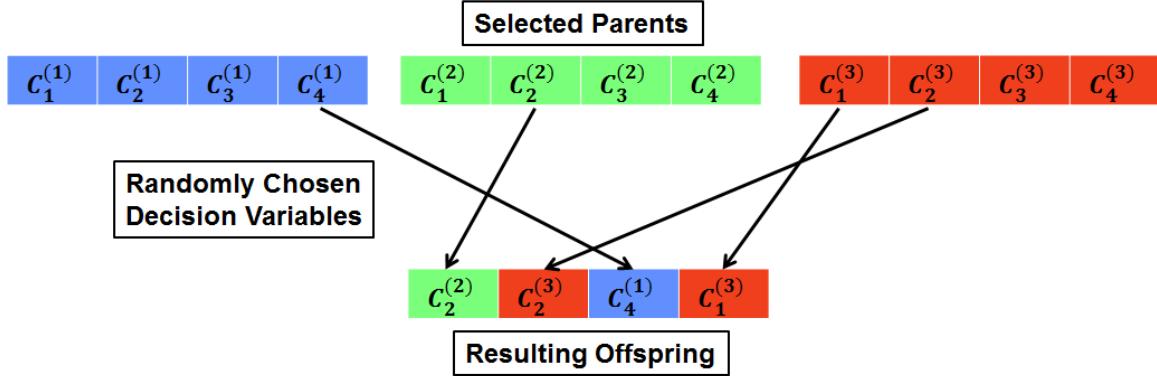


Figure 5.9. Dominant (discrete) recombination technique

5.10.2 Intermediate Recombination

Intermediate recombination is the first of three continuous recombination operators typically used in ESs. In intermediate recombination, the individual fitness values of the chosen subset of offspring vectors do not factor into the resulting combined parent vector $\bar{x}_{\frac{\mu}{\rho}}$ as illustrated in Equation (5.9) [151]. This is analogous to finding the centroid or geometric center of the space defined by the selected subset of ρ offspring vectors. This process can then be repeated μ times to generate the desired number of parent vectors.

$$\bar{x}_{\frac{\mu}{\rho}} = \frac{1}{\rho} \sum_{i=1}^{\rho} \bar{x}^{(i)} \quad (5.9)$$

5.10.3 Fitness-Weighted Recombination

If the intermediate recombination method is analogous to finding the geometric center of the selected ρ offspring vectors, then the fitness-weighted recombination technique is akin to finding the center of mass of the selected offspring. When using this method, the associated

fitness values that accompany each selected offspring vector determine how much relative weight the vector has in terms of producing the combined parent vector, $\bar{x}_{\frac{\mu}{\rho}}$ [187]. Equation (5.10) illustrates this process for a simple minimization problem where all fitness values are assumed to be greater than zero. It must be noted that, when using this technique, care is needed to appropriately scale fitness values when they have the potential to be equal to zero, or take on negative values.

$$\bar{x}_{\frac{\mu}{\rho}} = \frac{1}{\sum_{i=1}^{\rho} \frac{1}{F(\bar{x}^{(i)})}} \sum_{i=1}^{\rho} \frac{\bar{x}^{(i)}}{F(\bar{x}^{(i)})} \quad (5.10)$$

5.10.4 Fitness-Shaped Recombination

The next recombination technique is a special form of the weighted recombination method that reshapes the sampled fitness values based on logarithmic weights that are assigned to each chromosome. This method utilizes normalized logarithmic-shaped weights that are described in Equation (5.11) [189].

$$\begin{aligned} w_{i:\rho} &= \log\left(\rho + \frac{1}{2}\right) + \log(i) \\ \tilde{w}_{i:\rho} &= \frac{w_{i:\rho}}{\sum_{k=1}^{\rho} w_{k:\rho}} \\ s.t. \quad \sum_{i=1}^{\rho} \tilde{w}_{i:\rho} &= 1 \end{aligned} \quad (5.11)$$

After the normalized weights have been determined, a simple weighted average is applied to the top ρ selected offspring that have been sorted based on fitness to create a new parent vector, $\bar{x}_{\frac{\mu}{\rho}}$ as depicted in Equation (5.12). The sorted subset of offspring vectors and associated weights are described using the following notation: $(\bar{x}^{(i:\rho)})$ where i indicates the fitness-based rank out of ρ selected offspring. Using this technique, the best offspring vector will receive the largest weight value and the worst offspring that has been selected for recombination will receive the smallest weight value.

$$\bar{x}_{\frac{\mu}{\rho}} = \sum_{i=1}^{\rho} \tilde{w}_{i:\rho} \bar{x}^{(i:\rho)} \quad (5.12)$$

Figure 5.10 illustrates an example of the weight function for $\rho = 10$. This logarithmic weighting method is selected over a standard fitness weighted average to promote progress and prevent stagnation in cases where all parents have similar fitness values indicating very flat fitness landscapes [189].

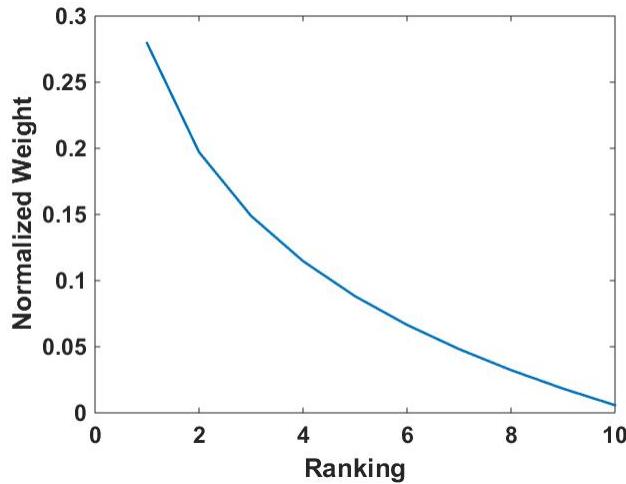


Figure 5.10. Logarithmic weighting scheme ($\rho = 10$)

5.11 Covariance Adaptation Techniques

Up to this point in the chapter, not much attention has been paid to the different techniques of evolving mutation distribution parameters. There has been a lot of work and development of increasingly sophisticated methods for evolving distribution covariances in ways that can significantly improve ES performance and convergence properties [50], [181]–[184].

5.11.1 One-Fifth Success Rule

The earliest attempt at evolving the covariance for a $(1 + 1)$ ES was initially developed by Rechenberg where the normal mutation scheme with a single σ for all decision variables (Equation (5.3)) was used [40]. Rechenberg developed this technique by determining the

optimal covariance success ratio, or percentage of generations that lead to an offspring with a better fitness value than the parent vector so that convergence speed is maximized for two different optimization problems. Rechenberg realized that there was an optimal range of σ values that could lead to better convergence speeds. As one might imagine, very large values of σ will likely have a lower success rate given that they will start to behave more like a random search across the fitness landscape and will have a smaller probability of finding an offspring that performs better than the current parent. On the other hand, a very small covariance will lead to a situation where the success rate will approach $\frac{1}{2}$ due to the locality of the search [79]. However, the issue here is that the successful steps will be very small and will result in slow convergence rates. Given a knowledge of these trades, Rechenberg analyzed success rates on two fairly simple cost functions. Equation (5.13) illustrates the first problem referred to as the corridor problem and Equation (5.14) depicts the second problem commonly referred to as the spherical cost function as it is used in much of ES theoretical analysis [49].

$$f_{\text{corridor}}(\bar{x}) = \begin{cases} x_1 & \text{if } |x_i| < 1 \text{ for } i = (2, \dots, n) \\ \infty & \text{otherwise} \end{cases} \quad (5.13)$$

$$f_{\text{sphere}}(\bar{x}) = \frac{1}{2} \sum_{i=1}^n x_i^2 \quad (5.14)$$

Rechenberg discovered that the optimal success rate for the corridor cost function was just over $\frac{1}{6}$ and a little less than $\frac{1}{3}$ for the spherical cost function. Given these results, the average success rate of $\frac{1}{5}$ was used to create the one-fifth success rule. In this scheme, σ is increased if the success rate is higher than $\frac{1}{5}$ and it is decreased when the success rate is less than $\frac{1}{5}$. The rate at which the covariance was increased or decreased in addition to the number of generations allowed before the success ratio is checked become user-defined parameters. Using the multiplicative scheme outlined in Equation (5.15) where the success rate is S_r , Schwefel determined reasonable values for the rate variable, ($0.85 \geq r < 1.0$) and proposed that the number of generations between each check should be equal to the problem dimension, given that the dimension is sufficiently large ($n > 30$) [180]. Lastly, the notation $\sigma^{(g)}$ indicates the current value of σ and $\sigma^{(g+1)}$ indicates the value that σ will

have for the next generation.

$$\sigma^{(g+1)} = \begin{cases} \sigma^{(g)} \times r & \text{if } S_r > \frac{1}{5} \\ \frac{\sigma^{(g)}}{r} & \text{if } S_r < \frac{1}{5} \\ \sigma^{(g)} & \text{if } S_r = \frac{1}{5} \end{cases} \quad (5.15)$$

$r < 1$

The one-fifth success rule was originally derived for the $(1 + 1)$ ES and has typically been employed by this type of ES. It can also be observed that it requires the assumption of symmetric, spherical distributions where each decision variable has the same covariance parameter. While this is a major step in optimizing the ES algorithms, it is somewhat limited given that it only tunes a single parameter as a response to a single measure.

5.11.2 Self-Adaptation

Self-adaption of endogenous strategy parameters essentially allows the covariance parameters to undergo mutation and recombination in a way that is similar to the other decision variables [79]. This idea was initially introduced by Schwefel in 1977 where each decision variable had a unique σ_i thus creating ellipsoidal distributions that were self-adapting [180]. This is the same as using a special case of the covariance matrix C that does not have any off-diagonal terms. Using this approach, each individual parent and offspring vector carries with it a unique set of strategy parameters $(\sigma_1, \dots, \sigma_n)$ resulting in each member having $2n$ decision variables that can be evolved. Using the afore-mentioned selection, mutation, and recombination operators, the covariance values that produce the best success rates should theoretically be the ones that survive and have the most influence on the population. This technique represents a fairly large step forward in terms of flexibility and robustness with regard to tuning the mutation distribution strategy parameters.

Upon implementation, there are several questions that need to be addressed regarding this method of self-adaptation. First of all, how does one prevent the evolution process from evolving these covariance parameters to negative values? To address this, a multiplicative approach for updating the covariance values is utilized. Second, how does one avoid an

endless cycle where an entire new set of strategy parameters is needed to describe the mutation process for the current set of strategy parameters? To address this concern, a constant global learning parameter, τ_0 , and a constant local learning parameter, τ , are introduced. In addition, only standard normal distributions with a mean of zero and covariance of one are used for this process. Equation (5.16) illustrates this case where each strategy parameter is mutated using a standard normal distribution coupled with a local learning parameter [180]. This entire vector is then perturbed once more by the product of the global learning parameter, τ_0 , and a global, normally distributed random value to produce a new set of strategy variables, $\bar{\sigma}_{new}$. In this method, an axis-parallel mutation scheme is used to mutate the decision variables.

$$\bar{\sigma}_{new} = \exp(\tau_0 \mathcal{N}_0(0, 1)) (\sigma_1 \exp(\tau \mathcal{N}_1(0, 1)), \dots, \sigma_n \exp(\tau \mathcal{N}_n(0, 1))) \quad (5.16)$$

Given that both the decision variables and the covariance vectors are perturbed using separate random vectors, there are a couple of issues that can degrade the ES effectiveness. One of the issues that can arise with this self-adaptation method is that selection noise and large fluctuations in the strategy parameters can occur and have a negative impact on ES performance [182]. These large fluctuations can occur in small populations given the nature of sampling from normal distributions where large steps may occur, even in distributions that have small covariance values. Along these same lines, a good update for the decision variables does not always indicate that the update in associated covariance was toward the optimal covariance value [182]. There are several techniques that can be done to help alleviate this issue. The simplest method is to ensure a sufficiently large parent population $\mu > 1.25n$ and implement intermediate recombination to help abate the selection noise that can occur [49]. Another technique is to use derandomized self-adaptation where the same random vector, \bar{z} , that is used to perturb the decision variables is also scaled and used to perturb the covariance values [182]. Ostermeier et al. illustrate that this method can minimize the selection noise and produce good results, even with small parent populations. Equation (5.17) illustrates an example of this technique where \circ is a component-wise product operator, k indicates the offspring number, $\mathbb{E}|\mathcal{N}(0, 1)| = \frac{\sqrt{2}}{\pi}$, d_1 and d are both user-defined constants [49].

$$\begin{aligned}
\bar{z}^{(k)} &= (\mathcal{N}_1(0, 1), \dots, \mathcal{N}_n(0, 1)) \\
\xi &= \tau \mathcal{N}(0, 1) \\
\bar{x}_{offspring}^{(k)} &= \bar{x}_{parent} + \exp(\xi) \bar{\sigma}^g \circ \bar{z}^{(k)} \\
\bar{\sigma}^{g+1} &= \bar{\sigma}^g \circ \exp^{\frac{1}{d_1}} \left(\frac{(|z_1^{(k)}|, \dots, |z_n^{(k)}|)^T}{\mathbb{E} |\mathcal{N}(0, 1)|} - 1 \right) \exp^{\frac{1}{d}}(\xi) \\
k &= (1, \dots, \lambda)
\end{aligned} \tag{5.17}$$

5.11.3 CSA-ES

Ostermeier et al. introduce the CSA-ES which adds a new aspect to the derandomized self-adaptation method where an evolution path vector, \bar{s}_σ , is used to track the successful mutation steps of the selected offspring [181]. This algorithm, in its original form, assumes axis-symmetric normal distributions, where the covariance matrix has no off-diagonal terms. In some ways, this algorithm has some similarities with the one-fifth success rule in that each covariance mutation is now able to account for how the entire population is progressing. By looking at a scaled history of the evolution path for the selected parent vectors over the generations, this algorithm is able to adapt the covariance parameters in a more meaningful way [181]. As an example, if the evolution path is oscillating back and forth in opposite directions, the covariance step sizes will decrease using this method. On the other hand, if the evolution path is continuing on in a given direction over multiple generations, this algorithm will increase the covariance step sizes so that the ES can progress more efficiently. Equation (5.18) illustrates the cumulative self-adaptation process for a $(\frac{1}{\rho}, \lambda)$ ES [49]. In this scheme: $c_\sigma = \sqrt{\rho/n+\rho}$, g is the generation number, d and d_1 are user-defined rates and are recommended to be $d = 1 + \sqrt{\rho/n}$ and $d_1 = 3n$. In addition, the notation $s_{\sigma,j}^{(g+1)}$ indicates the j th component of the updated evolution path vector $\bar{s}_\sigma^{(g+1)}$. Lastly, $\mathbb{E} \|\mathcal{N}(0, I)\|$ is the expectation of the Euclidean norm of the n -dimensional, normally distributed random vector $\mathcal{N}(0, I)$ with a zero mean and an n by n identity matrix for the covariance. This can be approximated as $\mathcal{N}(0, I) \approx \sqrt{n}$, an estimate that becomes increasingly accurate as the

search dimension, n , is increased [187].

$$\begin{aligned}
\bar{z}^{(k)} &= (\mathcal{N}_1(0, 1), \dots, \mathcal{N}_n(0, 1)) \\
\bar{x}_{offspring}^{(k)} &= \bar{x}_{parent} + \bar{\sigma}^{(g)} \circ \bar{z}^{(k)} \\
\bar{s}_{\sigma}^{(g+1)} &= \bar{s}_{\sigma}^{(g)}(1 - c_{\sigma}) + \sqrt{c_{\sigma}(2 - c_{\sigma})} \frac{\sqrt{\rho}}{\rho} \sum_{j=1}^{\rho} \bar{z}^{(j)} \\
\bar{\sigma}^{(g+1)} &= \bar{\sigma}^{(g)} \circ \exp^{\frac{1}{d_1}} \left(\frac{\left(|s_{\sigma,1}^{(g+1)}|, \dots, |s_{\sigma,n}^{(g+1)}| \right)^T}{\mathbb{E} |\mathcal{N}(0, 1)|} - 1 \right) \exp^{\frac{c_{\sigma}}{d}} \left(\frac{\sqrt{\sum_{j=1}^{\rho} \left(s_{\sigma,j}^{(g+1)} \right)^2}}{\mathbb{E} \|\mathcal{N}(0, I)\|} - 1 \right) \\
k &= (1, \dots, \lambda)
\end{aligned} \tag{5.18}$$

To help further illustrate how the evolution path vector is updated, the meaning of the factor $\sqrt{c_{\sigma}(2 - c_{\sigma})}$ is further investigated. It normalizes the mean perturbation vectors \bar{z} for the given distributions and was derived by Ostermeier et al. using the limit of the geometric series defined in Equation (5.19) below [181]. This factor diminishes the impact that any given generation has on the evolution path by approximately $1/3$ after every $1/c_{\sigma}$ generations so that the newest data has the largest effect on the covariance adaptation and older generations become increasingly less relevant as the algorithm progresses.

$$\lim_{m \rightarrow \infty} \sqrt{c_{\sigma}^2 + (c_{\sigma} (1 - c_{\sigma})^1)^2 + (c_{\sigma} (1 - c_{\sigma})^2)^2 + \dots + (c_{\sigma} (1 - c_{\sigma})^m)^2} = \sqrt{c_{\sigma}(2 - c_{\sigma})} \tag{5.19}$$

5.12 CMA-ES

The CMA-ES builds on the CSA-ES by increasing the amount of information that is used by the algorithm in an effort to improve convergence speed and accuracy. One of the primary differences between the two algorithms is that the CMA-ES evolves a full covariance matrix that can have off-diagonal terms allowing for correlated distributions that can rotate and can better fit a given fitness landscape [183]. This algorithm begins with a n-dimensional random vector as before, but applies it to the parent vector in a way that allows for the full covariance to be utilized to generate offspring, as described in Equation (5.20). This is

similar to Equation (5.6) except there is an additional step size control, $\sigma^{(g)}$, that allows the algorithm an additional scale factor to increase or decrease the step size for the covariance matrix. This step size will be evolved separately along with C in the CMA-ES.

$$\begin{aligned}\bar{z}^{(k)} &= (\mathcal{N}_1(0, 1), \dots, \mathcal{N}_n(0, 1)) \\ \bar{x}_{offspring}^{(k)} &= \bar{x}_{parent} + \sigma^{(g)} \sqrt{C^{(g)}} \bar{z}^{(k)}\end{aligned}\tag{5.20}$$

Given this offspring generation scheme, the CMA-ES then tracks two separate evolution paths according to Equation (5.21). Where \bar{s}_σ is the evolution path assuming no off-diagonal terms in the covariance, or the components of the evolution path that parallel each dimension. In addition, \bar{s}_C , tracks the full covariance evolution path, not including $\bar{\sigma}$. It can be seen that these equations are very similar to the evolution path calculation used for the CSA-ES, with a few subtle differences. The first change is that both evolution path equations leverage normalized fitness-shaped recombination weights \tilde{w}_k as described in Equation (5.11) above. This allows the best selected individuals and associated $\bar{z}^{(k)}$ vectors to have an impact on the evolution path that corresponds to their respective rank among the ρ selected chromosomes. The second change is that a new coefficient h_σ has been added to prevent the evolution path of the covariance matrix from growing too large and thus having too large of an impact on the shape of the distribution [49].

$$\begin{aligned}\bar{s}_\sigma^{(g+1)} &= \bar{s}_\sigma^{(g)}(1 - c_\sigma) + \sqrt{c_\sigma(2 - c_\sigma)} \sqrt{\frac{1}{\sum_{j=1}^{\rho} \tilde{w}_j^2} \sum_{j=1}^{\rho} \tilde{w}_j \bar{z}^{(j)}} \\ \bar{s}_C^{(g+1)} &= \bar{s}_C^{(g)}(1 - c_C) + h_\sigma \sqrt{c_C(2 - c_C)} \sqrt{\frac{1}{\sum_{j=1}^{\rho} \tilde{w}_j^2} \sum_{j=1}^{\rho} \tilde{w}_j \sqrt{C} \bar{z}^{(j)}} \\ h_\sigma &= \begin{cases} 1 & \text{if } \frac{\|\bar{s}_\sigma^{(g+1)}\|^2}{n} < \left(2 + \frac{4}{n+1}\right) \\ 0 & \text{otherwise} \end{cases}\end{aligned}\tag{5.21}$$

Next, the CMA-ES updates the mutation step factor, σ^{g+1} , the covariance matrix, C^{g+1} , and the parent vector mean, \bar{x}_{parent}^{g+1} according to Equation (5.22). It must be noted that in the

previously described $\left(\frac{1}{\rho}, \lambda\right)$ ESs, the parent vector mean could be updated using one of the continuous recombination techniques described in Sections 5.10.2-5.10.4. The CMA-ES uses a modified fitness-shaping recombination method that factors the covariance strategy parameters into the update of the parent vector as seen in Equation (5.22). [49]

$$\begin{aligned}
\sigma^{(g+1)} &= \sigma^{(g)} \exp^{c_\sigma / d} \left(\frac{\|\bar{s}_\sigma^{(g+1)}\|}{\mathbb{E} \|\mathcal{N}(0, I)\|} - 1 \right) \\
C^{(g+1)} &= (1 - c_1 + c_h - c_\rho) C^{(g)} + c_1 \left(\bar{s}_C^{(g+1)} \right) \left(\bar{s}_C^{(g+1)} \right)^T + c_\rho \sum_{j=1}^{\rho} \tilde{w}_j \sqrt{C^{(g)}} \bar{z}^{(k)} \left(\sqrt{C^{(g)}} \bar{z}^{(k)} \right)^T \\
\bar{x}_{parent}^{(g+1)} &= \bar{x}_{parent}^{(g)} + c_m \sigma^{(g+1)} \sqrt{C^{(g+1)}} \sum_{j=1}^{\rho} \tilde{w}_j \bar{z}^{(k)}
\end{aligned} \tag{5.22}$$

It can be seen that there are several coefficients that have been added to help control the behavior of this algorithm. Table 5.1 summarizes these coefficients along with values suggested by Hansen et al. [49]. A much more in-depth discussion and derivation of these coefficients along with CMA-ES open source code can be found in Hansen's tutorial on CMA-ES [187].

Symbol	Purpose	Suggested Value
\tilde{w}_j	Normalized fitness-shaped weights	See Sec 5.10.4
μ_ρ	Fitness weight normalizing factor	$\frac{1}{\sum_{j=1}^{\rho} w_j^2}$
c_σ	σ evolution path coefficient	$\frac{\mu_\rho}{n^2 + \mu_\rho}$
c_C	C evolution path coefficient	$\frac{4 + \frac{\mu_\rho}{n}}{n + 4 + \frac{2\mu_\rho}{n}}$
d	σ learning rate	$1 + \sqrt{\frac{\mu_\rho}{n}}$
c_1	C update coefficient	$\frac{2}{n^2 + \mu_\rho}$
c_ρ	C update coefficient	$\frac{\mu_\rho}{n^2 + \mu_\rho}$
c_h	C update coefficient	$c_1 (1 - h_\sigma^2) c_C (2 - c_C)$
c_m	\bar{x}_{parent} mutation coefficient	1.0

Table 5.1. Suggested Values For CMA-ES Coefficients

Lastly, it has been found that the CMA-ES has direct links and similar behavior characteristics to deterministic, gradient-based methods on certain classes of problems. More specifically, on simple convex functions such as the spherical cost function in Equation (5.14), it has been shown that the optimal covariance matrix found by the CMA-ES approaches the inverse of the Hessian matrix of the fitness function (I in the case of $f_{sphere}(\bar{x})$) [187].

5.13 NES

The NES is another algorithm that is able to evolve a full covariance matrix and account for correlations between decision variables. NESs, as introduced in 2008 by Wierstra et al., are algorithms that utilize random sampling to approximate a natural search gradient that is then used to update the search distribution parameters for the next generation [50]. The NES is able to approximate the gradient of the expected fitness w.r.t. distribution parameters by sampling a fitness landscape. It then applies this information to update the mean, \bar{x}_{parent} , and covariance, C , of the search in a way that moves and adjusts the associated distribution toward areas of higher expected fitness. This idea has a more straightforward and eloquent mathematical foundation than the CMA-ES. It also has links to gradient-based search methods which are commonly used to solve astrodynamical problems. In addition, this algorithm can still handle discontinuous search spaces and optimize problems using a simplistic, black-box approach just as the other evolutionary algorithms have done.

While the NES generally fits within the construct of a $\left(\frac{1}{\rho}, \lambda\right)$ ES with a single parent vector, it has a few differences that will become apparent as the algorithm is discussed in more detail. First off, ρ is typically equal to λ given that these algorithms use all of the sample points to estimate the search gradients. Given this, a sorting algorithm is used to assign fitness rank-based weights to all of the sample points using the logarithmic weighting scheme outlined in Section 5.10.4, but there is no need for a selection operator to truncate the population. In addition, the mean of the distribution, or \bar{x}_{parent} is not updated through a recombination process, but is instead updated from the estimated gradient information. Lastly, the covariance matrix is updated in a similar fashion as the distribution mean by leveraging an estimated fitness-based gradient relative to changes in covariance. While the NES is able to operate on a number of different search distributions, this section introduces the algorithm with the assumption that Gaussian distributions will be employed for sampling the search space.

5.13.1 Search Gradient

The foundation of the NES is based upon using a Monte Carlo sampling approach to sample the fitness landscape and use this information to approximate a search gradient. A search gradient is simply the gradient of the expected fitness with respect to the search distribution parameters (e.g. \bar{x}_{parent} and C for a Gaussian distribution). While there are several methods for approximating a search gradient, the original NES utilizes the gradient of a sample of the expected fitness combined with a log likelihood trick that was first introduced by Berny [190]. The density of a normal distribution is defined in Equation (5.23), where $\bar{x}_{offspring}$ is the sample point, \bar{x}_{parent} is the distribution mean, and C is the covariance matrix. In addition, n is the problem dimension and θ is a vector of distribution parameters (\bar{x}_{parent}, C) .

$$\pi(\bar{x}_{offspring}|\theta) = \frac{1}{(\sqrt{2\pi})^n \det(C)} \exp\left(-\frac{1}{2}(\bar{x}_{offspring} - \bar{x}_{parent})^T C^{-1} (\bar{x}_{offspring} - \bar{x}_{parent})\right) \quad (5.23)$$

With this definition, the fitness expectation $J(\theta)$ is defined in Equation (5.24).

$$J(\theta) = \mathbb{E}_\theta [f(\bar{x})] = \int f(\bar{x}) \pi(\bar{x}|\theta) d\bar{x} \quad (5.24)$$

Next, the gradient of the fitness expectation can be defined with Equation (5.25) using the log likelihood trick as described by Wierstra et al. [50]. The log likelihood trick essentially consists of multiplying the gradient in line two of Equation (5.25) by $\frac{\pi(\bar{x}|\theta)}{\pi(\bar{x}|\theta)}$ which is equivalent to one. Next, the 2nd and 3rd lines are equivalent given the definition of the derivative of a natural log $\left(\frac{d}{dx} \log(u) = \frac{1}{u} \cdot \frac{du}{dx}\right)$.

$$\begin{aligned}
\nabla_{\theta} J(\theta) &= \nabla_{\theta} \int f(\bar{x}) \pi(\bar{x}|\theta) d\bar{x} \\
&= \int f(\bar{x}) \nabla_{\theta} \pi(\bar{x}|\theta) d\bar{x} \\
&= \int f(\bar{x}) \nabla_{\theta} \pi(\bar{x}|\theta) \frac{\pi(\bar{x}|\theta)}{\pi(\bar{x}|\theta)} d\bar{x} \\
&= \int [f(\bar{x}) \nabla_{\theta} \log \pi(\bar{x}|\theta)] \pi(\bar{x}|\theta) d\bar{x} \\
&= \mathbb{E}_{\theta} [f(\bar{x}) \nabla_{\theta} \log \pi(\bar{x}|\theta)]
\end{aligned} \tag{5.25}$$

The search gradient can now be approximated from the random samples, \bar{x}_k with a simple summation as depicted in Equation (5.26) with λ being the number of random samples taken $\bar{x}_1, \dots, \bar{x}_{\lambda}$.

$$\nabla_{\theta} J(\theta) \approx \frac{1}{\lambda} \sum_{k=1}^{\lambda} f(\bar{x}_k) \nabla_{\theta} \log \pi(\bar{x}_k|\theta) \tag{5.26}$$

From here the logarithmic Gaussian density function and its gradients with respect to the Gaussian parameters are described in Equation (5.27).

$$\begin{aligned}
\log \pi(\bar{x}_{offspring}|\theta) &= -\frac{n}{2} \log(2\pi) - \frac{1}{2} \log(|C|) - \frac{1}{2} (\bar{x}_{offspring} - \bar{x}_{parent})^T C^{-1} (\bar{x}_{offspring} - \bar{x}_{parent}) \\
\nabla_{\bar{x}_{parent}} \log \pi(\bar{x}_{offspring}|\theta) &= C^{-1} (\bar{x}_{offspring} - \bar{x}_{parent}) \\
\nabla_C \log \pi(\bar{x}_{offspring}|\theta) &= \frac{1}{2} C^{-1} (\bar{x}_{offspring} - \bar{x}_{parent}) (\bar{x}_{offspring} - \bar{x}_{parent})^T C^{-1} - \frac{1}{2} C^{-1}
\end{aligned} \tag{5.27}$$

Applying Equations (5.26) and (5.27), it can be seen that an approximate search gradient can quickly be obtained from a set of λ Gaussian, random sample points that have associated fitness $f(\bar{x}_{offspring})$ values. Further studies in a later chapter will provide an in-depth investigation of how accurate these search gradients become as the number of sample points is increased.

5.13.2 Natural Search Gradient

With the fitness-based search gradient estimations w.r.t. both the mean and covariance, one more step is taken to increase the robustness and performance of the NES prior to updating \bar{x}_{parent} and C . Rather than using the gradient found in Equation (5.26), the natural gradient, $\tilde{\nabla}_\theta J(\theta)$, is calculated and used for the update due to a couple of reasons. First of, it is found that the standard search gradient can lead to slow convergence rates when encountering plateaus and ridges in fitness landscapes [41]. Second, the use of natural gradients can help address the oscillation that can occur on simple quadratic functions when the updates keep overshooting the optimal point and result in an infinite cycle where the optimal is never converged upon within a desired degree of accuracy [41]. The natural gradient addresses both of these issues by using a distance measure that is a better fit for the distance between distributions than a simple Euclidean distance. Equation (5.28) illustrates the technique used to convert the standard search gradient into a natural search gradient by using the Fischer information matrix, F .

The natural search gradient, $\tilde{\nabla}_\theta J(\theta)$, uses the inverse covariance to give more weight to gradient samples that have a low covariance and gives less weight to gradient samples that have a high covariance [50]. This, in essence, makes the natural search gradient more trustworthy than the standard search gradient in that it accounts for the covariance in a way that favors gradient samples that have lower variances over those that have higher variances. This concept of a natural gradient is explained in detail and further illustrated in Wierstra et al.'s papers [41], [50].

$$\begin{aligned} \tilde{\nabla}_\theta J(\theta) &= F^{-1} \nabla_\theta J(\theta) \\ \text{where: } & \\ F &= \mathbb{E} \left[\nabla_\theta \log \pi (\bar{x}_{offspring} | \theta) \nabla_\theta \log \pi (\bar{x}_{offspring} | \theta)^T \right] \end{aligned} \tag{5.28}$$

5.13.3 Incorporating Fitness-Shaped Weights

Weirstra et al. use a set of rank-based weights that are very similar to the weights discussed in Section 5.10.4 to shape the fitness landscape based on the rank-order of the offspring. [41]

This algorithm does not use selection, however, it does sort the offspring by fitness during each generation so that it can reassign these weights in a way that forces better offspring to always have larger weight factors than worse ones. While this likely decreases the accuracy of the search gradient in terms of the original fitness function, it has been shown to increase the convergence speed in areas of the fitness landscape that would otherwise have very shallow gradients. In addition, fitness-shaping creates a gradient that is invariant to rank-preserving transformations of the fitness function. In other words, as long as the fitness-based rank-order of the offspring is unchanged, the fitness function could have a very steep gradient or very shallow gradient and still appear the same after fitness shaping has been applied. This method is outlined in Equation (5.29) where $\hat{w}_{k:\lambda}$ indicates the rank-based weight for the k th best offspring vector $\bar{x}_{offspring,k:\lambda}$ out of λ total offspring. It can be noticed that the original fitness values, $f(\bar{x}_{offspring})$, do not directly appear in Equation (5.29). However, the associated fitness values do factor into the fitness-based ranking, or sorting, that determines what weight is associated with each offspring solution vector.

$$\begin{aligned}\hat{w}_{k:\lambda} &= \frac{\max \left\{0, \log \left(\frac{\lambda}{2} + 1\right) - \log(k)\right\}}{\sum_{j=0}^{\lambda} \max \left\{0, \log \left(\frac{\lambda}{2} + 1\right) - \log(j)\right\}} - \frac{1}{\lambda} \\ \nabla_{\theta} J(\theta) &\approx \sum_{k=1}^{\lambda} \hat{w}_{k:\lambda} \nabla_{\theta} \log \pi(\bar{x}_{offspring,k:\lambda} | \theta)\end{aligned}\quad (5.29)$$

Lastly, the complete NES method is summarized in Equation (5.30-5.32) after using fitness-shaping weights, natural gradients, and separating the equations for \bar{x}_{parent} and C . The algorithm begins by drawing λ random vectors of dimension n and applying them to the parent vector \bar{x}_{parent} or distribution mean as indicated in Equation (5.30).

$$\begin{aligned}\bar{z}^{(k)} &= (\mathcal{N}_1(0, 1), \dots, \mathcal{N}_n(0, 1)) \\ \bar{x}_{offspring}^{(k)} &= \bar{x}_{parent}^{(g)} + \sqrt{C^{(g)}} \bar{z}^{(k)} \\ k &= (1, \dots, \lambda)\end{aligned}\quad (5.30)$$

From here, each offspring vector is applied to the fitness function and they are sorted in terms of relative fitness achieved. After they have been sorted, the rank-based weights $\hat{w}_{k:\lambda}$ using the methodology described in Equation (5.29). Again, the notation $k : \lambda$ represents the k th best of λ total offspring in terms of fitness rank. Next, the search gradient of the expected fitness value relative to the mean, $\nabla_{\bar{x}_{parent}}$, and relative to the covariance, ∇_C , are determined using Equation (5.31).

$$\begin{aligned}\nabla_{\bar{x}_{parent}} J(\theta) &= \sum_{k=1}^{\lambda} \hat{w}_{k:\lambda} C^{-1} (\bar{x}_{offspring, k:\lambda} - \bar{x}_{parent}^{(g)}) \\ \nabla_C J(\theta) &= \sum_{k=1}^{\lambda} \hat{w}_{k:\lambda} \frac{1}{2} C^{-1} (\bar{x}_{offspring, k:\lambda} - \bar{x}_{parent}^{(g)}) (\bar{x}_{offspring, k:\lambda} - \bar{x}_{parent}^{(g)})^T C^{-1} - \frac{1}{2} C^{-1}\end{aligned}\tag{5.31}$$

After the search gradients have been determined, the next step is to calculate the Fischer information matrix for both $\bar{x}_{parent}^{(g)}$ and $C^{(g)}$ as outlined in Equation (5.32). With this information, the parent vector (distribution mean) and covariance matrix can be updated accordingly based on the natural gradients as indicated by Equation (5.32). In this technique, $\eta_{\bar{x}_{parent}}$ and η_C are constant learning rates defined by the user. Wierstra et al. recommend standard values for these rates, but they may require tuning on some problems [41].

$$\begin{aligned}F_{\bar{x}_{parent}}^{(g)} &= \frac{1}{\lambda} \sum_{k=1}^{\lambda} \left(C^{-1} (\bar{x}_{offspring, k:\lambda} - \bar{x}_{parent}^{(g)}) \right) \left(C^{-1} (\bar{x}_{offspring, k:\lambda} - \bar{x}_{parent}^{(g)}) \right)^T \\ F_{C^{(g)}} &= \frac{1}{\lambda} \sum_{k=1}^{\lambda} \left(\frac{1}{2} C^{-1} (\bar{x}_{offspring, k:\lambda} - \bar{x}_{parent}^{(g)}) (\bar{x}_{offspring, k:\lambda} - \bar{x}_{parent}^{(g)})^T C^{-1} - \frac{1}{2} C^{-1} \right) \\ &\quad \times \left(\frac{1}{2} C^{-1} (\bar{x}_{offspring, k:\lambda} - \bar{x}_{parent}^{(g)}) (\bar{x}_{offspring, k:\lambda} - \bar{x}_{parent}^{(g)})^T C^{-1} - \frac{1}{2} C^{-1} \right)^T \\ \bar{x}_{parent}^{(g+1)} &= \bar{x}_{parent}^{(g)} + \eta_{\bar{x}_{parent}} \left(F_{\bar{x}_{parent}}^{(g)} \right)^{-1} \nabla_{\bar{x}_{parent}} J(\theta) \\ C^{(g+1)} &= C^{(g)} + \eta_C \left(F_{C^{(g)}} \right)^{-1} \nabla_C J(\theta)\end{aligned}\tag{5.32}$$

The calculation of the search gradients, Fischer matrices, and updated parameters is an iterative process where the generation number g is incremented and the process repeats with a new set of offspring generated using the updated parameters (Equation (5.30)) and the algorithm continues to iterate until a user-defined stopping criteria is met.

5.14 Coordinate Invariant xNES

Seeking to make the NES more computationally efficient and completely invariant to linear transformations of the search space, Glasmachers et al. developed the xNES with several improvements and modifications over the original NES [191]. While the standard NES had some invariant characteristics with respect to linear search domain transformations due to its use of fitness-shaping, it was not completely invariant unless a limit on update step size were to be enforced. In addition, calculating the Fischer information matrix as well as determining its inverse proved to be some of the most computationally costly operations within the algorithm. Glasmachers et al. are able to address all of these shortcomings by utilizing an exponential transformation and working in the natural coordinate frame using an algorithm that they refer to as the xNES.

The first step to deriving this modified algorithm is to map the covariance matrix C to an exponential form using the matrix exponential. One way to do this is to use a UDU^T factorization as depicted in Equation (5.33) where U is an orthogonal matrix and D is a diagonal matrix. Furthermore, $\expm(M)$ is the matrix exponential of M and $\exp(D)$ is the element-wise exponential of the elements along the main diagonal of D . This results in the gradient updates to the exponential mapped covariance matrix $\expm(C) = C_\xi$ always producing valid covariance matrix that is invariant to linear transformations of the search domain [191].

$$\begin{aligned} M &= UDU^T \\ \expm(M) &= U \exp(D) U^T \end{aligned} \tag{5.33}$$

In addition to the exponential mapping of the C matrix, this algorithm does a transformation of the coordinate frame so that the original search distribution parameters are recovered

when the new, mapped mean and covariance parameters, (δ, M) are equal to zero. By utilizing this transformation into a natural coordinate frame, the natural gradient and the plain gradient are the same so that the Fischer information matrix becomes the identity matrix and thus trivial to calculate and invert. The complete xNES algorithm is now introduced without going into the specific coordinate mappings and derivations that can be found in the paper by Glasmachers et al. [191] and are further developed and investigated in Section 7.5.

The xNES works by evolving the distribution mean \bar{x}_{parent} , a step size σ , and a B matrix where the relation of the covariance parameters to the original covariance matrix are defined in Equation (5.34). It can be seen that this algorithm updates the covariance matrix by evolving both a step-size, σ , and a factorized covariance matrix, B , where $\sigma B = A$ and $AA^T = C$. This process shares some similarities with the CMA-ES in that it evolves both a step size and a matrix to adjust the covariance.

$$\begin{aligned} C &= AA^T \\ \sigma &= \sqrt[n]{|A|} \\ B &= \frac{A}{\sigma} \end{aligned} \tag{5.34}$$

As with the standard NES, this algorithm begins by drawing λ sample points from an n-dimensional normal distribution but uses a slightly different transformation to generate the offspring vectors $(\bar{x}_{offspring})$ as depicted in Equation (5.35).

$$\begin{aligned} \bar{z}^{(k)} &= (\mathcal{N}_1(0, 1), \dots, \mathcal{N}_n(0, 1)) \\ \bar{x}_{offspring}^{(k)} &= \bar{x}_{parent}^{(g)} + \sigma B \bar{z}^{(k)} \\ k &= (1, \dots, \lambda) \end{aligned} \tag{5.35}$$

At this point, the algorithm evaluates the fitness of the new set of offspring vectors. The xNES also uses the same fitness shaping scheme as the standard NES, outlined above, but

now calculates four separate search gradients as described in Equation (5.36). However, this algorithm introduces a set of new variables (δ, M) that are used to map the equations to a natural, exponential coordinate frame. Where $\nabla_\delta J(0, 0)$ is the search gradient of the expected fitness with respect to the mean in the newly mapped coordinate frame, and $\nabla_\sigma J(0, 0)$, $\nabla_M J(0, 0)$, and $\nabla_B J(0, 0)$ are gradients of the expected fitness evaluated at $(\delta = 0, M = 0)$ with respect to the covariance parameters in this new natural, exponential coordinate frame. This mapping of coordinate frames and xNES derivation is further developed and described in Section 7.5.

$$\begin{aligned}\nabla_\delta J(0, 0) &= \sum_{k=1}^{\lambda} \hat{w}_{k:\lambda} \bar{x}_{offspring, k:\lambda} \\ \nabla_M J(0, 0) &= \sum_{k=1}^{\lambda} \hat{w}_{k:\lambda} (\bar{x}_{offspring, k:\lambda} \bar{x}_{offspring, k:\lambda}^T - I) \\ \nabla_\sigma J(0, 0) &= \frac{\text{trace}(\nabla_M)}{n} \\ \nabla_B J(0, 0) &= \nabla_M J(0, 0) - \nabla_\sigma J(0, 0)I\end{aligned}\tag{5.36}$$

Given that these search gradients are already in the natural coordinate frame, the xNES is now able to update the search distribution using the update techniques described in Equation (5.37). As in the other algorithms, a new sample of offspring vectors are drawn from the updated distribution (Equation (5.35)) and this process continues until a user-defined stopping criteria is met.

$$\begin{aligned}\bar{x}_{parent}^{(g+1)} &= \bar{x}_{parent}^{(g)} + \eta_{\bar{x}_{parent}} \sigma^{(g)} B^{(g)} \nabla_\delta J(0, 0) \\ \sigma^{(g+1)} &= \sigma^{(g)} \exp\left(\frac{\eta_\sigma}{2} \nabla_\sigma J(0, 0)\right) \\ B^{(g+1)} &= B^{(g)} \exp\left(\frac{\eta_B}{2} \nabla_B J(0, 0)\right)\end{aligned}\tag{5.37}$$

It can be seen that this algorithm is more streamlined than the standard NES and does not require the calculation or inversion of Fischer information matrices. Lastly, Table 5.2

summarizes the suggested values for the learning parameters used in this algorithm [191].

Symbol	Purpose	Suggested Value
$\eta_{\bar{x}_{parent}}$	Learning rate for distribution mean	1
η_σ	Learning rate for covariance step size	$\frac{3(3+\log(n))}{5n\sqrt{n}}$
η_B	Learning rate for covariance matrix	$\frac{3(3+\log(n))}{5n\sqrt{n}}$
λ	Number of offspring	$4 + \lfloor 3 \log(n) \rfloor$

Table 5.2. xNES recommended parameters

5.15 ES Theory

Similar to GAs there are several overarching concepts that are typically referred to when describing ES convergence characteristics. At the highest level, there is the Evolutionary Progress principle (EPP) [151]. This principle simply states that the evolutionary progress of an ES type algorithm is the resultant sum of the process fitness gain and the process fitness loss that occurs from generation to generation as a result of the various operators internal to the algorithm [151]. Given this, the top level-goal for these algorithms is to design them in a way that maximizes the process fitness gain and minimizes the process fitness loss to increase the ES progress rate. Next, the idea of genetic repair (GR) is often used to illustrate how recombination operators can result in improved algorithm performance. While the schema theory and closely-related building block hypothesis focused on the potential for disruption of certain schema and the piecing together of different building blocks, GR concentrates on commonalities between parent vectors being passed from generation to generation. Beyer et al. use the idea of an intermediate recombination scheme to help illustrate how this idea of GR leads to convergence progress [180]. They look at each of the mutation step vectors $\bar{z}^{(k)}$ that result from the mutation operator and split them up into two separate vectors: one that leads to progress toward an optimal point, and one that leads to negative progress. As it turns out, due to selection, the ρ selected offspring used in intermediate recombination tend to have a strong correlation with regard to the positive progress vectors, and very little to no correlation with respect to the negative progress vectors. As a result, the negative progress vectors among the selected offspring vectors are almost canceled out during the intermediate recombination or averaging process. However, the positive progress vectors typically have some correlation and result in the updated centroid of the selected offspring

vectors moving in a direction that results in positive convergence progress [180].

5.15.1 Global Convergence Characteristics

The theoretical studies of ESs in regard to global convergence proofs and progress rates are in some ways similar to that of GAs in that they are limited to specific algorithm types and function classes. Convergence in terms of global aspects can be divided into converging to a globally optimal solution, and the closely related idea of time complexity or function evaluations required [151]. Global convergence for the elitist variants ‘+’ ESs has been proven mathematically for infinite time horizons in continuous spaces as summarized by Equation (5.38), where ϵ indicates a small vicinity around the global optimum, $F(\bar{x}^*)$. [79], [151] Looking at this from a practitioner’s point of view, this is of little use, given that their time is likely finite.

$$\lim_{g \rightarrow \infty} Pr \left\{ \left| F(\bar{x}^{(g)}) - F(\bar{x}^*) \right| \leq \epsilon \right\} = 1 \quad \epsilon > 0 \quad (5.38)$$

The ‘,’ ESs have not been proven to globally converge, even with infinite time horizons due to the fact that they can always leave an optimal solution without elitism being enforced [49]. In regards to time complexity, upper and lower bounds for simplified ES algorithms on certain classes of convex functions have been developed and mathematically proven. In fact it has been shown that ESs cannot achieve convergence rates faster than linear, $O(1/n)$, with constant λ and μ [49].

5.15.2 Local Convergence Characteristics

With these general results for global convergence being of little use to practitioners, local aspects of convergence have primarily been studied and developed for ESs by a number of authors with the hopes that general behavior traits increase understanding of how ESs will perform on real-world problems. Dynamic models typically use two metrics: local progress rate, ϕ and quality gain, \bar{Q} , as ways to measure performance of specific types of ESs on simple fitness functions [49], [151]. Quality gain is defined as the change in expected fitness of the aggregate population from one generation to the next [151]. Progress rate is the generational change in Euclidean distance between specific parent vectors and the optimal solution vector, \bar{x}^* , in parameter space. [151] As an example, the progress rate of

the $(1 + 1)$ ES on the spherical cost function is shown in Equation (5.39). [79] This was the metric that Rechenberg optimized to create the one-fifth success rule [180]. It can be seen that the progress rate for this algorithm on this specific fitness landscape is a function of the problem dimension, n , the Euclidean distance from the optimal point r , and the isotropic normal distribution standard deviation σ [79]. It is worth noting that both r and σ change from generation to generation and that erf is a standard error function.

$$\phi(n, r, \sigma) = \frac{\sigma}{\sqrt{2\pi}} \left(\exp \left(- \left(\frac{n\sigma}{\sqrt{8r}} \right)^2 \right) \right) - \frac{\sigma}{\sqrt{2\pi}} \left(\sqrt{\pi} \frac{n\sigma}{\sqrt{8r}} \left(1 - erf \left(\frac{n\sigma}{\sqrt{8r}} \right) \right) \right) \quad (5.39)$$

More recently, authors have been able to expand this type of analysis to various classes of convex quadratic functions, ridge functions, and constrained linear functions [49]. Furthermore, they have analyzed progress rates for $(1^+, \lambda)$ ESs, (μ^+, λ) ESs, and have investigated the benefits of recombination in the $(\mu/\rho^+, \lambda)$ ESs [151]. Unfortunately, isotropic normal distributions have always been used in the development of these models, leaving much work to be done regarding more modern ESs that use axis-parallel or even correlated normal mutation distributions (e.g. CMA-ES or NES) [49].

5.16 Conclusion

This chapter has introduced the reader to the basic types of ESs algorithms as they have been developed over the years. The standard selection, mutation, and recombination operators have been introduced and discussed in detail. Most importantly, this chapter has made the reader aware of the state-of-the-art in ESs by summarizing the CMA-ES, NES, and xNES algorithms. Lastly, this chapter has provided the reader with insight into the basic theoretical ideas and metrics that can be used to predict ES performance on limited classes of fitness functions. All of this material creates an important foundation that this dissertation will now build upon to introduce new ideas and techniques that will further advance the current state and, in some ways, change the standard way of thinking with regard to ESs.

CHAPTER 6: Unscented Sampling In ESs

6.1 Introduction

The use of Monte Carlo concepts to sample the domain of objective functions has been standard practice in ES algorithms over the years [180]. This random sampling approach often uses Gaussian distributions to emulate biological mutation and fulfill all of Beyer's requirements for a well-suited ES mutation operator [151], as discussed in Section 5.8. The difficulty with any Monte Carlo type approach, is that a fairly large number of samples are typically required to obtain accurate statistics. The issue that comes with larger sample sizes is the increased computational cost in terms of objective function evaluations. This chapter investigates the idea of using deterministically chosen sampling points as a new approach to address this challenge. The application of two different deterministic approaches are investigated in this chapter, with the primary focus being on sigma points [192] given their ability to efficiently parameterize Gaussian distributions in a way that increases the effectiveness of sampling the objective function landscapes. This idea becomes even more valuable as the dimension of the problem of interest is increased because the number of sigma points scales linearly with dimension. The reduction in the number of sample points also reduces the cost associated with evaluating a high dimensional objective function that could potentially be expensive in terms of required computation time. This is of particular interest in the world of optimal control where the numerical methods of solving optimal control problems often require optimization over a fairly high dimensional search space [30]. This proposed technique of deterministic sampling in evolutionary optimization thus holds the potential to be an extremely valuable contribution when applying these types of heuristic search algorithms to optimal control problems.

In the pursuit of an algorithm that could have better global search characteristics and the potential to work well with deterministic sampling points, this chapter begins by proposing a new ES covariance adaptation scheme that draws is loosely based on concepts from simulated annealing algorithms. Next, this new ES is modified to leverage deterministically chosen sigma points within its mutation operator. The chapter then introduces a challenging

set of highly-multimodal test problems from literature which are used as a benchmark from which to measure the effectiveness of the algorithms being compared. From here, trades are conducted to empirically and theoretically influence the design of these new ESs in terms of recombination types and recombination fraction sizes, ρ . In order to determine the effectiveness of deterministic sampling, the sigma point ES is compared with an ES that is identical in all regards except it uses random sampling. This chapter further characterizes the unscented ES by testing its ability to find locally optimal solutions by leveraging gradient information from the benchmark test problems. From here, the deterministic ES is compared with a state-of-the-art CMA-ES to further illustrate its performance on the benchmark functions. The chapter then briefly explores the use of Gauss-Hermite points using Smolyak sparse grids as an alternative way of selecting deterministic sampling points for some of the more challenging problems. Finally, the new ES is applied to solve the discretized lunar lander problem using parallel computation.

6.2 random simulated annealing evolution strategy (RSA-ES)

The RSA-ES is a new kind of $(^1/\rho, \lambda)$ ES that uses a standard, Monte Carlo approach where the λ offspring are chosen from an n-dimensional, axis-parallel Gaussian distribution, as described in Equation (5.4) and used to form a single parent vector ($\mu = 1$). It utilizes a fitness rank-based truncation selection technique to select the subset of ρ offspring from the set of sample points. The specific recombination technique along with the value of ρ are analyzed and selected following a set of trade studies conducted in Section 6.5. The primary difference between the RSA-ES and a standard ES is its covariance adaptation method.

Avoiding convergence to a sub-optimal, local minimum/maximum solution can often be difficult when considering highly multimodal problems [33]. The RSA-ES, USA-ES, and GHSA-ES algorithms all utilize a simulated annealing inspired covariance adaptation scheme that is designed to enhance global search characteristics while increasing the potential for ESs to work well with unscented sampling techniques. The goal is to allow for a more controlled transition between exploring the global problem space and exploiting specific regions that have above average fitness traits. In order to achieve these results, the RSA-ES employs a simulated annealing inspired covariance matrix adaptation technique. This method creates a tunable search processes with a balanced approach that effectively transitions from exploration to exploitation of the search space in a predictable and re-

peatable manner. This is desirable when looking at highly multimodal problems where the initial exploration phase (large covariance) should enable the algorithm to consider the entire search domain and the exploitation phase (small covariance) should be able to narrow in on the local optima. This not only proves to be effective in obtaining global solutions on a number of the test problems, but is also repeatable allowing for more controlled experiments and the isolation of variables for more meaningful comparisons.

This covariance adaptation technique utilizes a “cooling schedule” that shares some similarities with a simulated annealing algorithm to adjust the covariance matrix over time, rather than conventional methods that allow the algorithm to evolve the covariance using Gaussian processes. The RSA-ES is designed to begin with a fairly large covariance in each dimension during the initial generations of the run. After each generation, the Cholesky factor of the covariance matrix, A , decays according to an exponential cooling schedule that is given in Equation (6.1). In this It can be seen that A is a function of the problem dimension, n , and it decays exponentially as the generation number, g , increases. It must be noted that an axis-parallel mutation scheme is assumed for the experiments in this chapter, however, this need not be the case. This cooling schedule seems to work well for the problems tested empirically in this chapter, but may require further tuning to optimize performance on a specific problem. Furthermore, a number of different cooling functions could potentially be used, depending on the characteristics of the problems being solved. The specifics of the recombination method used within this algorithm will be empirically investigated and designed based on the results later in Section 6.5.

$$A_{i,i}^{(g+1)} = \frac{A_{i,i}^{(0)}}{\left(1 + \frac{1}{n}\right)^g} \quad \forall i = [1, \dots, n] \quad (6.1)$$

6.3 USA-ES

The USA-ES is identical to the RSA-ES with its only difference being how the offspring are generated within the mutation operator. While the RSA-ES uses a conventional approach of random sampling from a parent-centric Gaussian distribution, the USA-ES utilizes 3rd order sigma points to efficiently and strategically sample the search space and generate offspring for a given parent vector.

6.3.1 Unscented Sigma Point Parameterization

Monte Carlo sampling can generate fairly accurate results with large enough sample sizes, however, this approach becomes very costly given that larger sample sizes directly lead to more required function evaluations and computational time durations. This aspect is increasingly important as the problem dimension grows and the computation of objective functions become more expensive. In the search for more efficient sampling techniques that circumvent some of these issues, sigma points quickly became appealing for this application. Julier et al. initially introduced the concept of an unscented transform and the associated sigma points in the context of nonlinear filtering [192]. This idea has been further developed and applied in a number of ways in filtering problems by Julier and others over the years [193]–[196]. The general form of 3rd order unscented sigma points as described by Julier [197] is illustrated in Equation (6.2). This set of points is referred to as 3rd order because they offer the minimum number of points that are able to exactly match up to 3rd order statistical moments for a given Gaussian distribution. For these calculations κ is a user defined parameter that affects the spread of the sigma points which are defined by the covariance matrix, C , and the problem dimension or distribution dimension, n . Using this method, χ is a n by $2n + 1$ matrix, where each column defines a unique point in the n -dimensional space. It must also be noted that this form of the sigma point calculation initially finds the 3rd order sigma points, χ , for a normal distribution with zero mean and an identity matrix for the covariance.

$$\chi = \left[\begin{array}{c|ccccccccc} 0 & \sqrt{(n + \kappa)} & 0 & 0 & \cdots & -\sqrt{(n + \kappa)} & 0 & 0 & \cdots \\ 0 & 0 & \sqrt{(n + \kappa)} & 0 & \cdots & 0 & -\sqrt{(n + \kappa)} & 0 & \cdots \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \sqrt{(n + \kappa)} & 0 & 0 & \cdots & -\sqrt{(n + \kappa)} \end{array} \right]$$

with $\kappa \in \mathbb{R}^n$

(6.2)

Given the set of sigma points for a standard normal distribution outlined in Equation (6.2), an affine transformation can be used to find $\bar{\chi}_i$ which is the set of sigma points that have been translated by a given mean μ and scaled by the square root of the covariance \sqrt{C} so

that any normal distribution can be represented, as depicted in Equation (6.3).

$$\bar{\chi}_i = \mu + \sqrt{C} \chi_i \quad i = [1, \dots, 2n + 1] \quad (6.3)$$

Using Julier's moment matching concept, 3rd order sigma points are calculated as depicted in Equation (6.2). These points may be utilized as sample points in the ES in order to evaluate the fitness landscape in the neighborhood of the parent. The idea is to generate a more accurate sampling of the fitness function than would be obtained by even a fairly large number of randomly selected points. The effectiveness of this idea as compared to random sampling is illustrated during the method comparisons throughout this chapter. One of the greatest benefits of using these 3rd-order sigma points versus another parametrization is that with sigma points, the number of points only grows linearly as the dimension is increased. Standard interpolating points such as Gauss-Hermite encounter an exponential growth in the number of points with respect to the number of dimensions through the use of tensor products to populate an n -dimensional mesh. This can be somewhat mitigated by using Smolyak [198] sparse grid representation techniques, however, the curse of dimensionality can still manifest itself at higher dimensions, especially when the accuracy order and problem dimension are increased. This idea of using Smolyak sparse grids will be further investigated in Section 6.9. An algorithm that can be applied to higher dimensional problems is of great interest, especially when considering optimal control applications where problems can require discretization in both time and parameter space resulting in high dimensional search spaces.

It must be noted that the κ value in Equation (6.2) essentially defines the spread, or concentration of the sigma points about the mean. As depicted in Figure 6.1, a larger value of κ results in points that are more spread out than points resulting from smaller values of κ . Negative values of κ are possible, but care should be taken to make sure that they do not exceed the dimension n , as this will produce imaginary numbers.

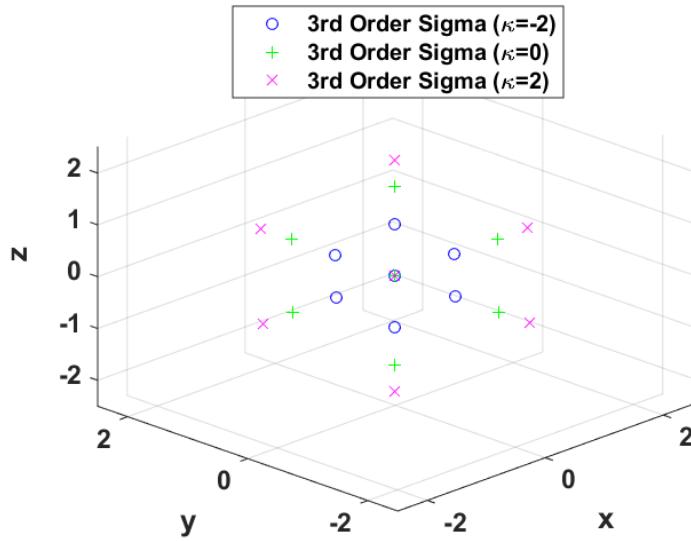


Figure 6.1. Illustration of 3-D 3rd order sigma points with varying κ values

A mutation operator that utilizes deterministic sampling with sigma points is not without its own unique challenges that must be overcome. Figure 6.2 illustrates that there are zones in the tails of the Gaussian that will not be sampled due to the locations of the finite number of points with respect to the bell-curve. This exclusion of the distribution tails may limit exploration. However, the simulated annealing covariance adaptation technique described above, coupled with large initial covariance values can have the ability to overcome this issue by altering the covariance of these points as the algorithm progresses. This directly coincides with Beyer's requirement of reachability for a well-suited mutation operator [151]. Using the modified covariance scheme with sufficiently large initial standard deviations and random starting points, this mutation operator can satisfy all of Beyer's requirements: scalability, absence of bias, symmetry, and reachability as they are presented in Section 5.8 [151]. With this, the deterministic points are symmetric about the mean of the distribution satisfying the requirement of symmetrical mutation. Lastly, the choice of a parameterized Gaussian distribution satisfies the requirement for absence of bias, without any specific knowledge of the search space.

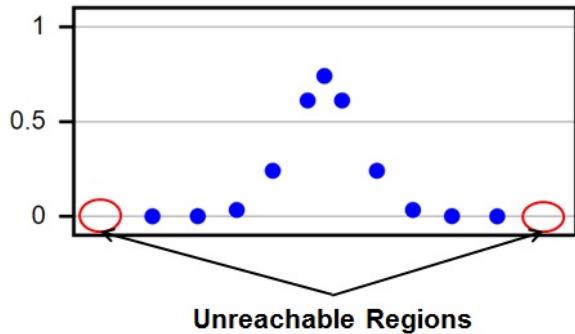


Figure 6.2. Unreachable zones with parameterization of Gaussian

Like the RSA-ES, the specific recombination operator and selection size ρ will be further investigated and developed in the following sections. However, it can be seen that this is a fairly simplistic ES algorithm with unique characteristics that give it the potential to work well on high-dimensional problems. It must be noted that the USA-ES and the other deterministic sampling ESs investigated in this chapter are different from ESs that utilize random sampling in that they are repeatable. In other words, they will always take an identical evolution path, given the same set of user-defined parameters and initial starting point for a given problem from run to run of the algorithm. Of course measures can be taken to add additional aspects of randomness to these algorithms, if this characteristic is desired. Random initial starting points, can often times be enough to fully explore the search space with multiple restarts. In addition, the use of parallel implementations can be used, as will be seen in Section 6.11.

6.4 Benchmark Test Problems

Table 6.1 summarizes a set of very challenging and highly multimodal test problems has been put together by Hansen et al. and initially used to test the effectiveness of their state-of-the-art CMA-ES [189]. This is a set of problems that have appeared throughout ES literature and have been deemed difficult for standard ESs to solve for one reason or another. In addition to being challenging to solve, this set of problems is easily scalable to any desired number of dimensions, n . This can be very useful for analysis purposes when developing an understanding of how each algorithm performs as the dimension of the search domain is increased. It can also be seen that a non-symmetric initialization range is defined

for each problem forcing the starting point to lie within a certain range. This is done to help prevent the RSA-ES and USA-ES from exploiting problem symmetry characteristics and to make the resulting experiments more representative of optimization search capability.

Problem	Objective Function	Init. Range
Ackley	$f_{Ackley}(\bar{x}) = 20 - 20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^n x_i^2}\right)$ $+ e - \exp\left(\frac{1}{n}\sum_{i=1}^n \cos(2\pi x_i)\right)$	$[1, 30]^n$
Bohachevsky	$f_{Bohachevsky}(\bar{x}) = \sum_{i=1}^{n-1} \left[x_i^2 + 2x_{i+1}^2 - 0.3\cos(3\pi x_i) \right. \\ \left. - 0.4\cos(4\pi x_{i+1}) + 0.7 \right]$	$[1, 15]^n$
Griewank	$f_{Griewank}(\bar{x}) = \frac{1}{4000} \sum_{i=1}^n x_i^2 - \prod_{i=1}^n \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	$[10, 600]^n$
Rastrigin	$f_{Rastrigin}(\bar{x}) = 10n + \sum_{i=1}^n \left(x_i^2 - 10\cos(2\pi x_i) \right)$	$[1, 5]^n$
Scaled Rastrigin	$f_{RastScaled}(\bar{x}) = 10n + \sum_{i=1}^n \left((10^{\frac{i-1}{n-1}} x_i)^2 - 10\cos(2\pi 10^{\frac{i-1}{n-1}} x_i) \right)$	$[1, 5]^n$
Skewed Rastrigin	$f_{RastSkew}(\bar{x}) = 10n + \sum_{i=1}^n \left(y_i^2 - 10\cos(2\pi y_i) \right)$ <p style="text-align: center;">with $y_i = \begin{cases} 10x_i & \text{if } x_i > 0 \\ x_i & \text{otherwise} \end{cases}$</p>	$[1, 5]^n$
Schaffer	$f_{Schaffer}(\bar{x}) = \sum_{i=1}^{n-1} \left(x_i^2 + x_{i+1}^2 \right)^{0.25} \left[\sin^2 \left(50(x_i^2 + x_{i+1}^2)^{0.1} \right) + 1.0 \right]$	$[10, 100]^n$
Schwefel	$f_{Schwefel}(\bar{x}) = 418.9828872724339n - \sum_{i=1}^n x_i \sin \left(\sqrt{ x_i } \right)$	$[-500, 300]^n$

Table 6.1. Multimodal objective functions for benchmark problem set

All of the test functions in Table 6.1 have global optimum cost values equal to zero at the point $\bar{x}^* = [0]^n$ with the Schwefel function being the exception ($\bar{x}_{Schwefel}^* = [420.96874636]^n$) with a global optimum cost value that is also equal to zero.

Figure 6.3 illustrates the complicated fitness landscapes of the benchmark problem suite in two dimensions. It can be seen that these functions are all highly multimodal and have the potential to be fairly difficult in terms of finding the globally optimal point.

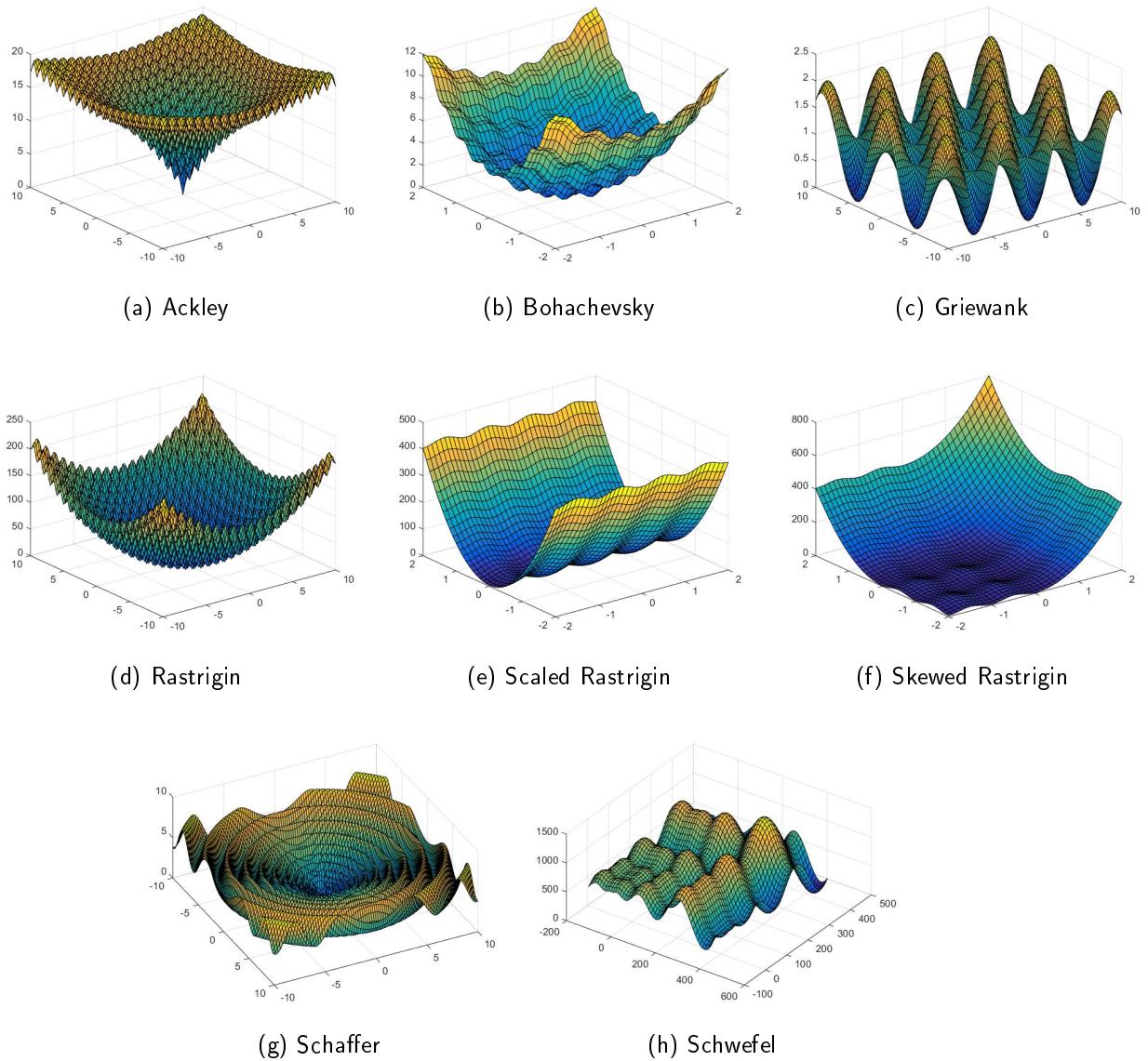


Figure 6.3. 3-D Visualization Of Benchmark Problems

6.5 Recombination Trade Studies

Before investigating how these ESs perform relative to each other, a series of experiments regarding the recombination methods are first conducted to help optimize the design of the recombination operator used within these algorithms. This helps ensure that the design of

the USA-ES and RSA-ES is well-suited for the modified covariance adaptation scheme. In addition, the USA-ES utilizes a new deterministic sampling scheme that also warrants some experimentation with regard to the recombination techniques used.

6.5.1 Recombination Operator Type

The first set of tests applies the USA-ES and RSA-ES algorithms to several of the problems from Table 6.1 comparing three different recombination techniques. The first recombination method investigated is the standard, intermediate recombination operator where the ρ selected offspring, $\bar{x}^{(i)}$, are equally weighted having an equal contribution to the parent vector, $\bar{x}_{\frac{\mu}{\rho}}$, as outlined in Section 5.10.2. Next, a fitness-weighted recombination method that is similar to the one described in Section 5.10.3 is investigated. It has been slightly modified with a simple check to ensure that no division by a zero occurs. If one of the fitness values is within machine precision of zero, a small positive term is added to it so that the solution vector is still appropriately weighted but the division by zero is avoided. This modified version of the fitness-weighted recombination technique is outlined in Equation (6.4). Where eps is a MATLAB variable that is set to machine precision ($eps = 2.2204 \times 10^{-16}$). In addition $F(\bar{x}^{(i)})$ is the fitness function being evaluated and $S(F(\bar{x}^{(i)}))$ is a simple check function that prevents a division by zero.

$$\begin{aligned}\bar{x}_{\frac{\mu}{\rho}} &= \frac{1}{\sum_{i=1}^{\rho} \frac{1}{S(F(\bar{x}^{(i)}))}} \sum_{i=1}^{\rho} \frac{\bar{x}^{(i)}}{S(F(\bar{x}^{(i)}))} \\ S(F(\bar{x}^{(i)})) &= \begin{cases} F(\bar{x}^{(i)}) & \text{if } F(\bar{x}^{(i)}) \geq eps \\ eps & \text{if } F(\bar{x}^{(i)}) < eps \end{cases}\end{aligned}\tag{6.4}$$

Lastly, the logarithmic fitness shaping technique that is described in Section 5.10.4 is used as the third recombination method in this set of experiments.

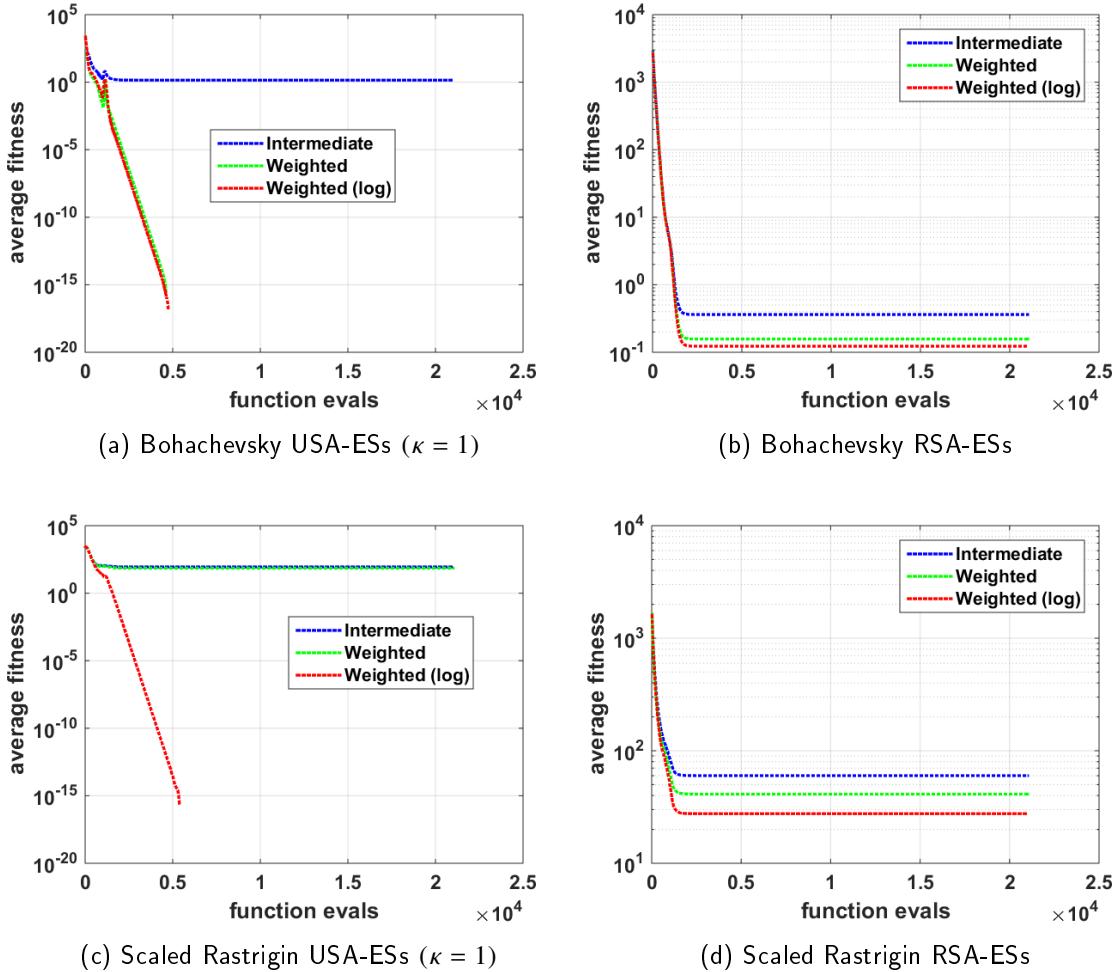


Figure 6.4. Recombination method comparison on “shallow” 10-D functions

This experiment runs each ES variant 100 times and averages the fitness evolution over all of the trials for each problem. In addition, each algorithm is allowed to run for 1,000 generations for each trial with an initial standard deviation in each dimension equal to half of the initialization range for each problem. Lastly, this set of experiments uses the top performing half of the offspring with, $\lambda = 2n + 1$, to generate the parent vector, $\rho = \lfloor \lambda/2 \rfloor$. The size of λ will be increased for the RSA-ES in later sections, however, it is set to the same value as the USA-ES for the purposes of determining appropriate recombination techniques. Figure 6.4 illustrates the trades between the various recombination techniques on both the USA-ES and RSA-ES variants. In both the Bohachevsky and Scaled Rastrigin

problems, the fitness landscape has regions where the fitness gradients are shallow and relative fitness values have the potential to be fairly close in value among the various sample points (see Figures 6.3b and e). In these types of problems, it can be seen that the logarithmic fitness weighting performs better than the other methods (Figure 6.4). This is because standard fitness weighted recombination technique can stagnate with very slow update progress when the fitness values of the selected offspring are relatively close in value. However, the logarithmic shaping idea does not depend on the actual fitness values and only uses the fitness-based rank of the selected offspring to create the new parent vector. Given this, the resulting parent vectors formed by this technique are invariant to the relative differences in fitness values between the chosen offspring vectors, as long as their respective order remains the same. The intermediate recombination operator is also invariant to this closeness in relative fitness, as it does not account for fitness in the construction of the new parent vector. While this allows the methods using intermediate recombination to continue to make progress on shallow fitness landscapes, it does not always result in updates to the parent vector that are in the best direction regarding fitness improvements given that neither fitness, or the fitness-based rank factor into the creation of the new parent vector. This intermediate recombination method solely relies on the truncation selection to improve the average fitness over the progression of the algorithm.

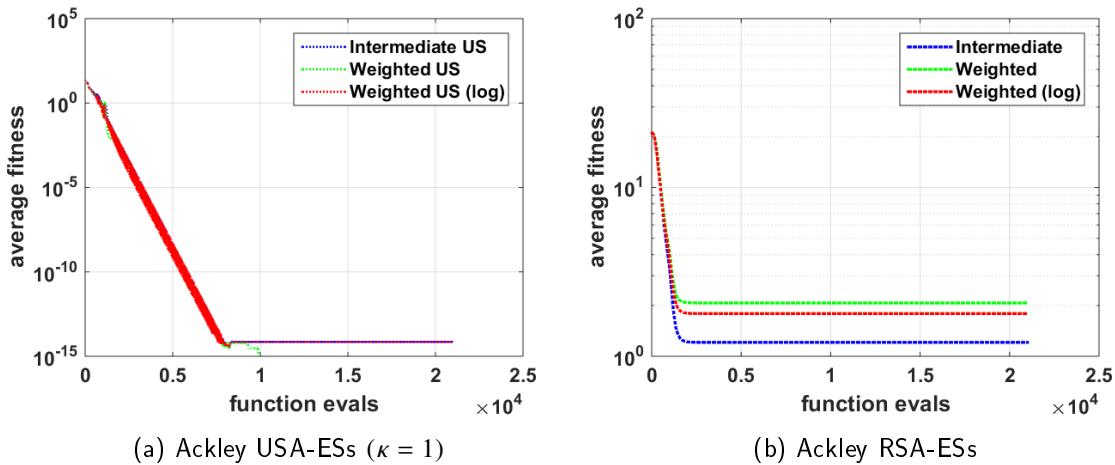


Figure 6.5. Recombination method comparison on “steep” 10-D function

Figure 6.5 illustrates the same set of tests only on the Ackley function which has steeper

fitness gradients where there are fewer regions where the relative fitness between sample points will be close in value (see Figure 6.3a). As can be seen, the logarithmic weighted recombination technique is not significantly better than the standard fitness weighted recombination technique. In this case, the RSA-ES achieves better performance with the intermediate recombination operator. One of the reasons for this, might be that the Ackley functions has a fairly well-defined global search direction, or direction toward the global optimal. Given that these tests only use the top half of the offspring for recombination as a result of the selection operator, the intermediate recombination technique is able to make fairly accurate moves in regard to fitness, despite not directly factoring fitness into the production of the parent vector. In addition, the relatively small sample size of $\rho = \lfloor \lambda / 2 \rfloor$ where $\lambda = 21$ for a 10D problem, may cause a scenario where the intermediate recombination is able to take larger steps and make more efficient updates than the weighted recombination methods, as seen in Figure 6.5b.

After running these tests on all of the functions in Table 6.1, it is empirically determined that the logarithmic weighted recombination method performs the best for the majority of the problems for both the USA-ES and RSA-ES. Given this, it is chosen as the recombination operator and is utilized for the experiments in the remainder of this chapter.

6.5.2 ρ Size

The next set of trades investigates the size of the selected offspring fraction, ρ , to be chosen from the set of offspring by the selection operator. This sub-group is then used by the recombination operator to create the next parent vector. Based on the recombination method comparisons in the previous section, the logarithmic recombination operator is used for these experiments. The experiment setup is similar with the recombination method trades in the previous section in that each algorithm undergoes 100 runs for each problem and is allowed to evolve for 1,000 generations. Both algorithms (USA-ES/RSA-ES) use the same number of offspring where $\lambda = 21$ for the 10 dimensional versions of the test functions outlined in Table 6.1. As will be seen later in this chapter, the RSA-ES does better with larger sample sizes, given that it uses Monte Carlo techniques. The difference between this set of trials is that a comparison between three different selected sub-group sizes is conducted where: the single best offspring is chosen ($\rho = 1$), the top half of the offspring are selected ($\rho = \lambda / 2$), and all of the offspring are used for recombination ($\rho = \lambda$) so that ρ

takes on three different values.

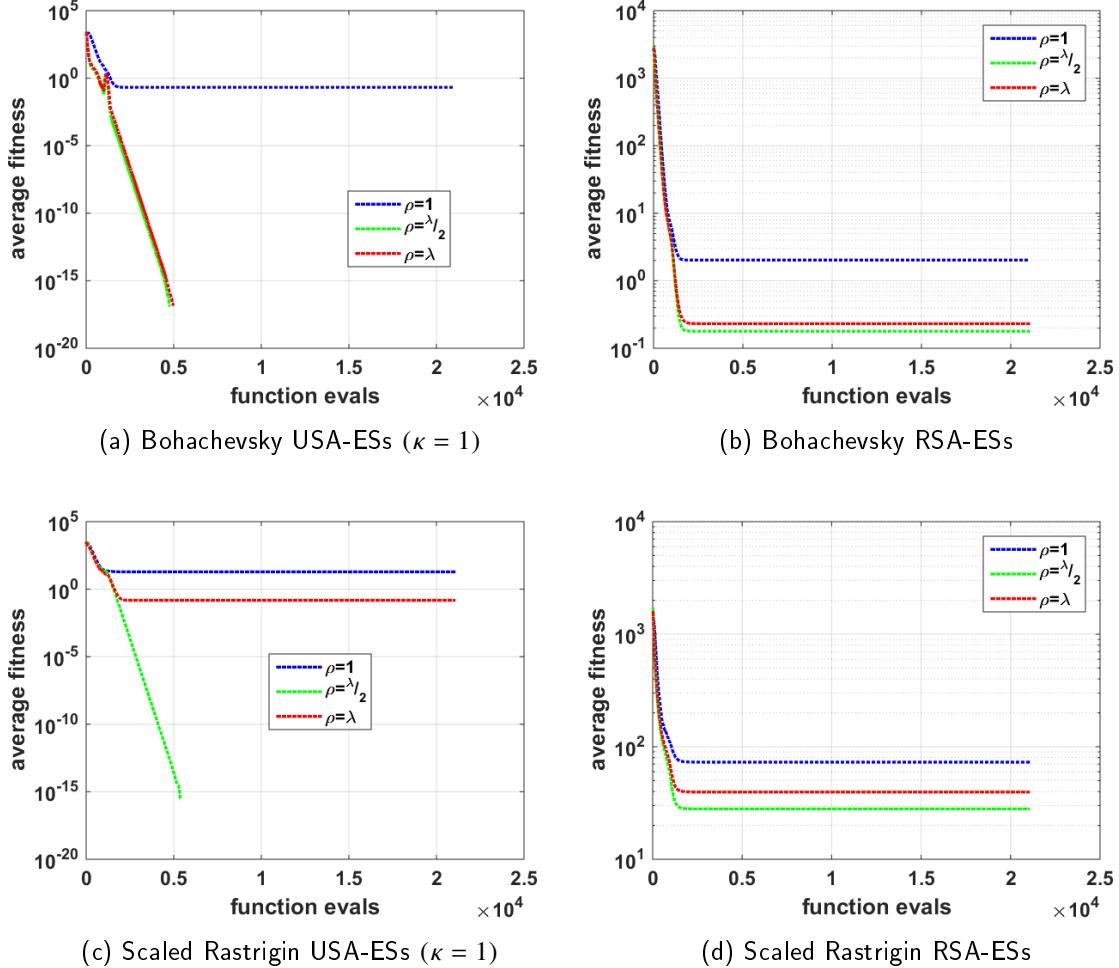


Figure 6.6. ρ comparison on “shallow” 10-D functions

Figure 6.6 depicts the results for these trade studies on the same “shallow” fitness functions comparing how the various ρ values affect both the USA-ES and the RSA-ES. As recommended by Hansen for his CMA-ES, setting $\rho = \lambda / 2$ seems to perform fairly well on these functions [49], [187]. Given that the logarithmic weighting scheme is applied in both of these algorithms, the lower performing half of the selected group of offspring would have weights that are fairly similar in value given the nature of the logarithmic curve. With this the best performing solutions should still be favored. However, using the entire

set of offspring for recombination may introduce some error into the update direction, or slow progress by incorporating poorly performing offspring. As an example, imagine the case where the last half of the offspring are in the complete opposite direction than one would want the algorithm to progress in. By including them in the recombination process, the parent vector might progress more slowly than it would have if only the top half of the offspring were selected for recombination. This seems to be the case with most of the problems, but not all of them, indicating that there are times when it is useful to consider larger portions of the population for recombination. In fact, Figure 6.7 illustrates the results for the Ackley function where it can be seen that the RSA-ES actually performs best when using the entire offspring population for recombination. In these cases, some error might be preferable to help induce exploration and prevent getting stuck in locally optimal points. Lastly, the Schwefel function was the only one where choosing a single point for recombination performed best for both the USA-ES and RSA-ES. This is likely due to the fact that the Schwefel function has a large initialization range, making it potentially desirable to take large steps from one generation to the next. By only utilizing the single best offspring vector as the parent, the algorithm has the potential to make larger steps in the search domain than would be likely for the same algorithm using the average of multiple samples. However, for most problems, too much information is lost by only utilizing the single best offspring in the creation of a parent vector.

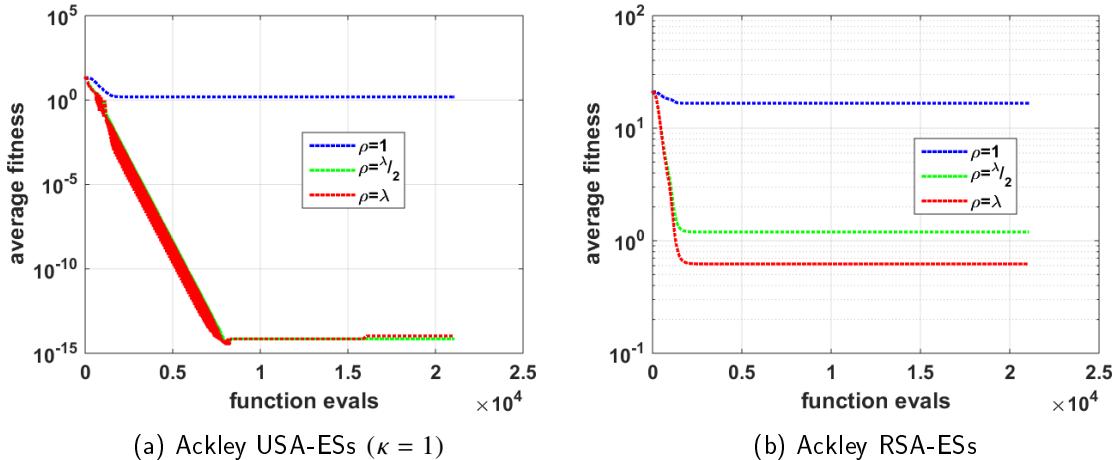


Figure 6.7. ρ comparison on “steep” 10D function

For the USA-ES using the setting of $\rho = \lambda / 2$ results in the best performance on six of the eight functions tested in 10-D. For the RSA-ES this same ρ setting performs best on five of the eight problems. This seems to indicate that this setting is somewhat problem-dependent; however, given the results from this section, the setting of $\rho = \lambda / 2$ will be used for the remainder of this chapter, given that it results in the best performance for both ES algorithms on the majority of the problems tested for the 10-D case.

With the appropriate trade studies for recombination types and ρ sizes complete, Algorithm 1 illustrates how the RSA-ES operates accounting for all of the steps where $A^T A = C$. For the remainder of the problems, the diagonal terms of the A matrix are set equal to half of the initialization range for each benchmark test problem.

Algorithm 1: RSA-ES Algorithm

Initialize: $\bar{x}_{parent}^{(0)} = \text{random uniform } \in [LB, UB]^{\mathbb{R}^n}$;

User-Defined Parameters: g_{max} , λ , ρ , and $A^{(0)}$;

$g = 0$;

Define Logarithmic Weights::

$$w_{i:\rho} = \log\left(\rho + \frac{1}{2}\right) + \log(i) \quad \forall i = [1, \rho];$$

$$\tilde{w}_{i:\rho} = \frac{w_{i:\rho}}{\sum_{k=1}^{\rho} w_{k:\rho}};$$

while stopping criteria not satisfied **do**

Obtain λ normal random n-dimensional sample points::

$$\bar{z}^{(k)} = \mathcal{N}(0, I);$$

Transform sample points::

$$\bar{x}_{offspring}^{(k)} = \bar{x}_{parent}^{(g)} + A^{(g)} \bar{z}^{(k)} \quad \forall k = [1, \lambda];$$

$$\text{Evaluate Fitness } F\left(\bar{x}_{offspring}^{(k)}\right) \quad \forall k = [1, \lambda];$$

Sort $\bar{x}_{offspring}^{(k)}$ w.r.t. fitness values;

Find New Parent Vector using the top ρ selected offspring::

$$\bar{x}_{parent}^{(g+1)} = \sum_{i=1}^{\rho} \tilde{w}_{i:\rho} \left[\bar{x}_{offspring} \right]_{i:\rho};$$

Update Covariance Factor::

$$A^{(g+1)} = \frac{A^{(0)}}{\left(1 + \frac{1}{n}\right)^g};$$

$$g = g + 1;$$

end

Algorithm 2 illustrates the USA-ES. It can be seen that it is almost identical, the only difference between the two methods is the use of deterministic sampling to create new offspring instead of Monte Carlo sampling.

Algorithm 2: USA-ES Algorithm

Initialize: $\bar{x}_{parent}^{(0)}$ = random uniform $\in [LB, UB]^{\mathbb{R}^n}$;

User-Defined Parameters: g_{max} , ρ , and $A^{(0)}$;

Use unscented sample points for χ_k ;

$g = 0$;

Define Logarithmic Weights::

$$w_{i:\rho} = \log\left(\rho + \frac{1}{2}\right) + \log(i) \quad \forall i = [1, \rho];$$

$$\tilde{w}_{i:\rho} = \frac{w_{i:\rho}}{\sum_{k=1}^{\rho} w_{k:\rho}};$$

while stopping criteria not satisfied **do**

$$\bar{\chi}_k = \bar{x}_{parent}^{(g)} + A^{(g)} \chi_k \quad \forall k = [1, \lambda];$$

$$\text{Evaluate Fitness } F(\bar{\chi}_k) \quad \forall k = [1, \lambda];$$

Sort $\bar{\chi}_k$ w.r.t. fitness values;

Find New Parent Vector using the top ρ selected offspring::

$$\bar{x}_{parent}^{(g+1)} = \sum_{i=1}^{\rho} \tilde{w}_{i:\rho} \bar{\chi}_{i:\rho};$$

Update Covariance Factor::

$$A^{(g+1)} = \frac{A^{(0)}}{\left(1 + \frac{1}{n}\right)^g};$$

$$g = g + 1;$$

end

6.6 USA-ES vs. RSA-ES Comparison

Now that some simple trades have been conducted and the algorithms have been designed based on those results, a comparison between the USA-ES and RSA-ES is warranted. The results outlined in Tables 6.2-6.4 were obtained by running the two developed ESs (RSA-ES and USA-ES) with the same number of iterations, parents, and offspring. Each of these ESs is run 100 times using 1,000 iterations per run ($g_{final} = 1000$). An average fitness progression is obtained by averaging the results from each of the 100 runs. By averaging the fitness progression over all of the runs, a measurement that is similar to average quality gain (as defined in Section 5.15.2) is observed in the fitness domain. Engineers are typically

more interested in the best fitness, or objective function value found, and not necessarily as interested in the progress rate in terms of parameter space. For this reason, the experiments in this chapter primarily focus on the average fitness progression. It must be noted that in many of these experiments, the algorithms may find the global optimal for a number of the trials, but consistently finding the global optimum or values that are close in fitness is what results in the best average fitness progressions. The average fitness progressions that end up at zero, or within machine precision of zero, hit the global optimum solution for all of the 100 trials. In addition, 99 % confidence intervals are calculated using Equation (6.5) with a z-score ($Z = 2.58$) to help illustrate the run-to-run variation in results. Each of these cases uses the set number of offspring that is defined by the number of sigma points generated in addition to the distribution mean (or parent) ($\lambda = 2n + 1$). This number of offspring is likely too small in many cases for Monte Carlo sampling used within the RSA-ES. With this, each ES also utilizes the same number of parents ($\mu = \left\lfloor \frac{\lambda}{2} \right\rfloor$) for these simulations. This is done to create a baseline of how well the USA-ES does in optimizing each function using the sigma points and is later treated as a target value for the next set of experiments. For each test function the global optimum is $f(\bar{x}^*) = 0$. In addition, the initial standard deviations in each dimension are set to the distance between upper bound and lower bound on the initialization ranges for each of the problems in Table 6.1. Lastly, the value of κ is set to $\kappa = 1$ for the experiments in this chapter.

$$CI = \bar{x} \pm \frac{Z\sigma}{\sqrt{\# \text{ of trials}}} \quad (6.5)$$

Function	$f(\bar{x})$ Evals	USA $f_{avg\ best}(\bar{x})$ 99% CI	RSA $f_{avg\ best}(\bar{x})$ 99% CI
Ackley	21,000	$1.1 \times 10^{-14} \pm 0.0$	1.5768 ± 0.4386
Bohachevsky	21,000	$9.2 \times 10^{-16} \pm 1.2 \times 10^{-17}$	0.1827 ± 0.0282
Griewank	21,000	0.0219 ± 0.0013	$0.0065 \pm 4.7 \times 10^{-4}$
Rastrigin	21,000	$3.4 \times 10^{-14} \pm 9.4 \times 10^{-16}$	8.0592 ± 0.2617
Scaled Rastrigin	21,000	$6.9 \times 10^{-14} \pm 3.0 \times 10^{-16}$	26.3193 ± 1.4650
Skewed Rastrigin	21,000	8.1985 ± 0.1125	23.6601 ± 0.6189
Schaffer	21,000	$3.6 \times 10^{-20} \pm 5.6 \times 10^{-22}$	2.3084 ± 0.2449
Schwefel	21,000	770.0818 ± 26.4045	$2,137.7 \pm 29.6259$

Table 6.2. 10-D USA-ES vs. RSA-ES results ($\rho = 10$ $\lambda = 21$)

Function	$f(\bar{x})$ Eval	USA $f_{avg\ best}(\bar{x})$ 99% CI	RSA $f_{avg\ best}(\bar{x})$ 99% CI
Ackley	41,000	$7.7 \times 10^{-15} \pm 1.7 \times 10^{-16}$	0.3993 ± 0.2292
Bohachevsky	41,000	0 ± 0	0.0676 ± 0.0176
Griewank	41,000	0.0234 ± 0.0015	0 ± 0
Rastrigin	41,000	$7.4 \times 10^{-15} \pm 1.2 \times 10^{-15}$	11.7803 ± 0.2898
Scaled Rastrigin	41,000	$1.4 \times 10^{-16} \pm 1.2 \times 10^{-16}$	35.4859 ± 1.5514
Skewed Rastrigin	41,000	7.5119 ± 0.1571	37.1219 ± 0.5996
Schaffer	41,000	$0.0012 \pm 4.9 \times 10^{-4}$	0.1791 ± 0.0482
Schwefel	41,000	631.8764 ± 8.7571	4563.6 ± 35.0276

Table 6.3. 20-D USA-ES vs. RSA-ES results ($\rho = 20 \lambda = 41$)

Function	$f(\bar{x})$ Eval	USA $f_{avg\ best}(\bar{x})$ 99% CI	RSA $f_{avg\ best}(\bar{x})$ 99% CI
Ackley	81,000	$4.1 \times 10^{-10} \pm 7.7 \times 10^{-12}$	$1.7 \times 10^{-9} \pm 8.0 \times 10^{-12}$
Bohachevsky	81,000	0 ± 0	0.0212 ± 0.0076
Griewank	81,000	0.0213 ± 0.0019	0 ± 0
Rastrigin	81,000	$6.0 \times 10^{-14} \pm 2.3 \times 10^{-15}$	17.9292 ± 0.3419
Scaled Rastrigin	81,000	$8.9 \times 10^{-15} \pm 1.1 \times 10^{-15}$	38.7151 ± 0.9424
Skewed Rastrigin	81,000	3.1142 ± 0.1043	58.6130 ± 0.6119
Schaffer	81,000	0.0069 ± 0.0012	0.0263 ± 0.0019
Schwefel	81,000	$1,305.0 \pm 1.7716$	$9,550.6 \pm 49.2329$

Table 6.4. 40-D USA-ES vs. RSA-ES results ($\rho = 40 \lambda = 81$)

The results of Tables 6.2-6.4 illustrate sigma point sampling is more effective than random sampling when the number of sampling points are equal, which is to be expected. Figure 6.8 illustrates a comparison of the average fitness progressions for several of the 40-D cases. This gives a better illustration of what is being portrayed in 6.2-6.4 and illustrates convergence (or reaching the global optimum within machine precision) in terms of the average fitness evolution path for both algorithms. It can also be seen that the USA-ES fitness progression seems to have some oscillation, or cycling in it as it converges on a solution in some of these problems. This is made possible due to the fact that this ES is a ‘,’ implementation (parents are not considered in selection), making it possible for the parent vector to get worse in terms of fitness from one generation to the next. In addition, the cooling schedule for the covariance in Equation (6.1) results in a covariance “cooling” rate that decreases as the problem dimension is increased. With this, the covariance remains larger for longer periods of time in the higher dimensional problems, so this effect is more

noticeable in the higher dimensional problems. The underlying cause for this oscillation is the fact that the USA-ES uses the same set of points for each generation, with the exception of them being scaled by the covariance matrix and shifted by the new distribution mean that results from the fitness shaped recombination technique. The noise in the figures is caused by the recombination operator overshooting the optimal point in an opposite direction with each generation. This cyclical approach is occurring in the USA-ES but not in the RSA-ES due to the deterministic, unchanging nature of the points which in turn makes the recombination operator behave in a similar way from one generation to the next. This overshoot is improved along with the average fitnesses as the covariance decreases, forcing the overshoot to get smaller and the fitness values to approach that of the global/local optimum it is trending toward. This further illustrates the importance of an appropriate covariance adaptation scheme.

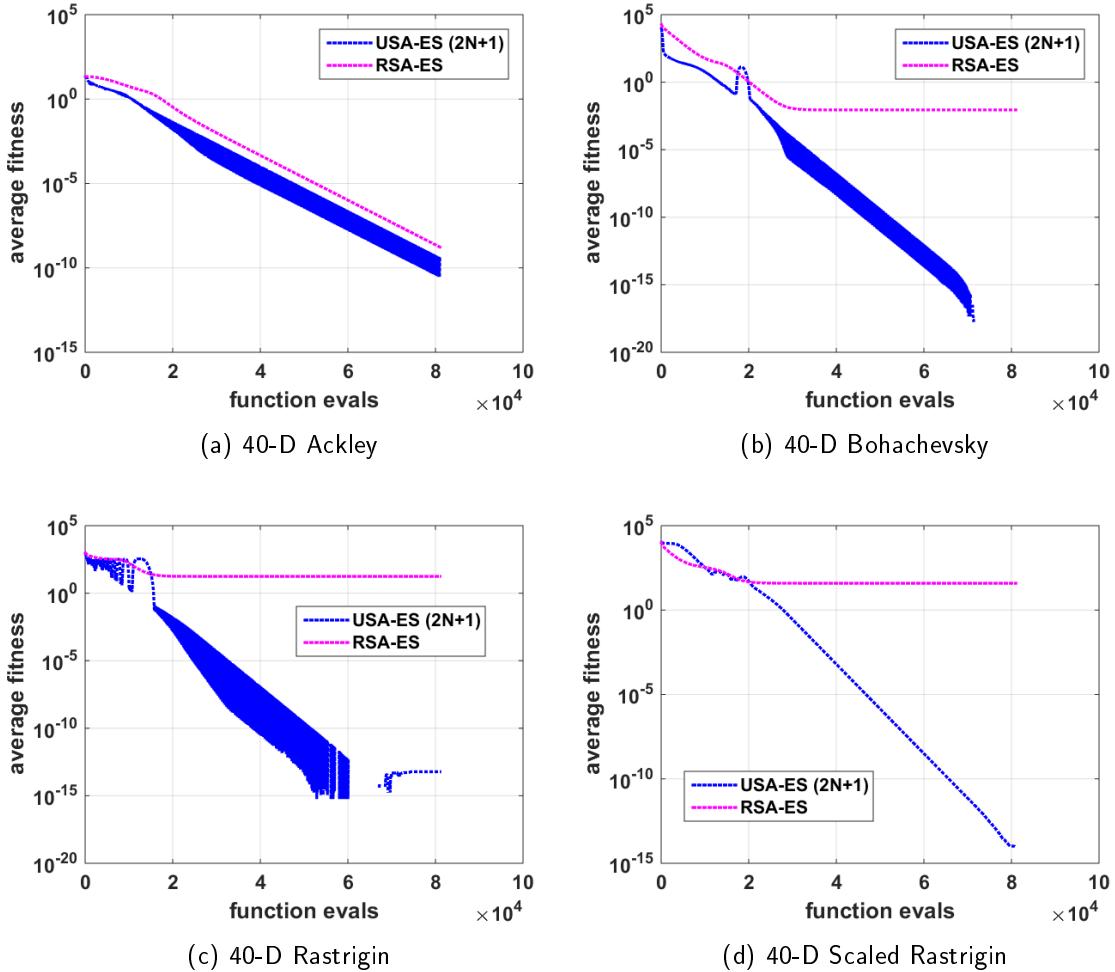


Figure 6.8. Average USA-ES/RSA-ES fitness progression on 40-D test functions with equal number of sampling points

Lastly Table 6.5 illustrates the percentage of runs that result in a solution that produced a fitness value that is less than 10^{-8} . This value is used as a tolerance to account for computational error, and is considered to be reasonably close to the globally optimal fitness for these problems. This indicated how often each algorithm was able to obtain a globally optimal solution during the 100 trials. It can be seen that the results illustrate similar trends as were seen above in Tables 6.2-6.4.

Function	USA % Success	RSA % Success
Ackley 10-D	100	93
Ackley 20-D	100	98
Ackley 40-D	100	100
Bohachevsky 10-D	100	71
Bohachevsky 20-D	100	88
Bohachevsky 40-D	100	98
Griewank 10-D	10	60
Griewank 20-D	12	100
Griewank 40-D	17	100
Rastrigin 10-D	100	0
Rastrigin 20-D	100	0
Rastrigin 40-D	100	0
Scaled Rastrigin 10-D	100	0
Scaled Rastrigin 20-D	100	0
Scaled Rastrigin 40-D	100	0
Skewed Rastrigin 10-D	0	0
Skewed Rastrigin 20-D	0	0
Skewed Rastrigin 40-D	1	0
Schaffer 10-D	100	3
Schaffer 20-D	96	6
Schaffer 40-D	0	0
Schwefel 10-D	3	0
Schwefel 20-D	0	0
Schwefel 40-D	0	0

Table 6.5. USA-ES vs. RSA-ES success probabilities

6.6.1 USA-ES/RSA-ES Solution Quality Study

Next, a second set of tests is performed where the intent is to determine how many random sample points are required to match the solution quality of the sigma point based approach. In order to achieve this each algorithm is run for 1,000 generations. This is repeated 100 times and the average best fitness is determined for each method. If the RSA-ES does not achieve a solution that has an average fitness value that is within a tolerance of 10^{-6} of the USA-ES, the number of sample points is increased by ten. This tolerance is used to help prevent longer experiment run times when the algorithms are sufficiently close in terms of solution quality. This process of incrementally increasing λ for the RSA-ES is repeated until the RSA-ES solution quality is as good, or better than the USA-ES. It must

be noted that another way to increase the number of function evaluations would be to hold the number of points constant and increase the number of generations. However, after some experimentation, it is observed that one-thousand generations is sufficient for both algorithms to converge to a solution and stabilize. Given this, the best way to perform this experiment is to fix the number of generations to 1,000 for both methods and simply increase the number of sampling points. Lastly, an upper limit on the number of points (1,501) is used to prevent infinite loops from occurring in this exercise.

Referring to Tables 6.6-6.8, it can be seen that there are a number of problems where the RSA-ES requires significantly larger sampling sizes and thus longer run times to achieve similar results as the USA-ES. In fact, it can be seen that the $2n + 1$ USA-ES becomes more and more favorable as the problem dimension is increased in terms of the number of problems that the RSA-ES is able to match performance of the USA-ES. As the number of dimensions is increased to 40, there are five problems that the RSA-ES is unable to match the performance of the USA-ES after using up to 1,501 sampling points where the USA only requires 81 sigma points. On the problems where both algorithms eventually obtain the same solution quality, the runtime ratio of RSA-ES to USA-ES gives another metric that helps indicate how much more efficient the USA-ES is on these problems. A ratio is utilized due to the fact that computational times will vary from computer to computer, however the ratio between the two should remain fairly constant, despite varying hardware configurations.

Function	USA-ES $f(\bar{x})$ Eval	RSA-ES $f(\bar{x})$ Eval	Runtime $\frac{RSA}{USA}$
Ackley	21,000	21,000	1.11
Bohachevsky	21,000	91,000	6.79
Griewank	21,000	21,000	1.39
Rastrigin	21,000	631,000	42.83
Scaled Rastrigin	21,000	1,501,000 (limit)	104.76
Skewed Rastrigin	21,000	391,000	28.98
Schaffer	21,000	1,501,000 (limit)	106.89
Schwefel	21,000	1,501,000 (limit)	79.71

Table 6.6. USA-ES/RSA-ES 10D solution quality results

Function	USA-ES $f(\bar{x})$ Eval	RSA-ES $f(\bar{x})$ Eval	Runtime $\frac{RSA}{USA}$
Ackley	41,000	41,000	1.21
Bohachevsky	41,000	81,000	3.92
Griewank	41,000	41,000	1.69
Rastrigin	41,000	891,000	39.64
Scaled Rastrigin	41,000	1,501,000 (limit)	64.23
Skewed Rastrigin	41,000	1,501,000 (limit)	75.50
Schaffer	41,000	1,501,000 (limit)	62.70
Schwefel	41,000	1,501,000 (limit)	45.10

Table 6.7. USA-ES/RSA-ES 20D solution quality results

Function	USA-ES $f(\bar{x})$ Eval	RSA-ES $f(\bar{x})$ Eval	Runtime $\frac{RSA}{USA}$
Ackley	81,000	91,000	1.47
Bohachevsky	81,000	141,000	3.58
Griewank	81,000	81,000	2.34
Rastrigin	81,000	1,501,000 (limit)	52.98
Scaled Rastrigin	81,000	1,501,000 (limit)	40.71
Skewed Rastrigin	81,000	1,501,000 (limit)	58.79
Schaffer	81,000	1,501,000 (limit)	37.75
Schwefel	81,000	1,501,000 (limit)	27.44

Table 6.8. USA-ES/RSA-ES 40D solution quality results

It can be seen that there are some cases where the RSA-ES is able to match the same solution quality as the USA-ES and others where it cannot before hitting the upper limit of sample points. Figure 6.9 illustrates one of each cases for the 40-D test functions. It can be seen that the RSA-ES only needs a few more sample points to match the performance of the USA-ES, where increasing the number of sample points to 1,501 still does not significantly improve the fitness on the Scaled Rastrigin function.

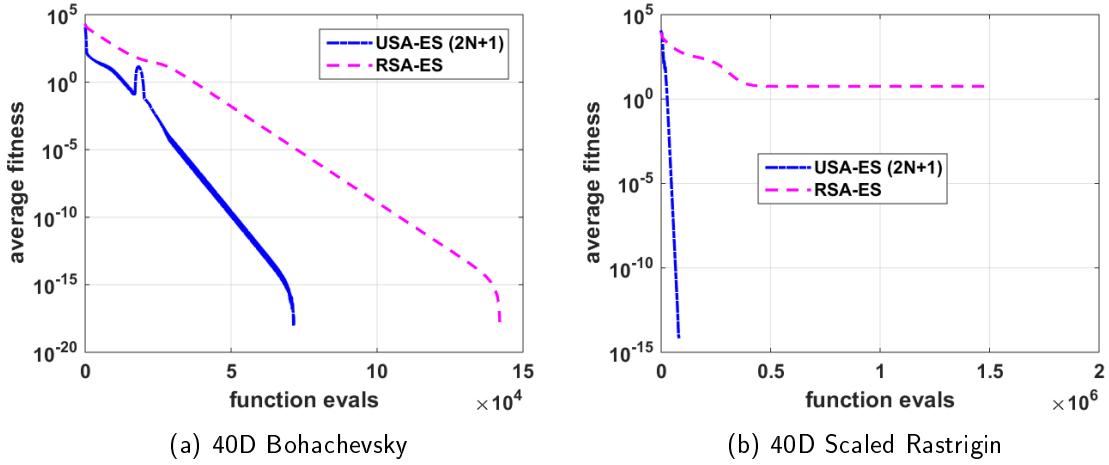


Figure 6.9. Average USA-ES/RSA-ES fitness progression as sample points and function evaluations are increased

Figure 6.10 provides a good illustration of how dimensionality affects each algorithm when looking at the fitness or objective function value progression. It can be seen that the USA-ES is almost unaffected by increasing the problem dimension while the RSA-ES quickly runs into issues and has difficulty as the dimension is increased. This is a very important benefit of using the proposed unscented sampling technique instead of conventional Monte Carlo techniques that are typically used. Again, this becomes increasingly important for discretized optimal control problems that can have a large number of decision variables given required resolutions in space and time.

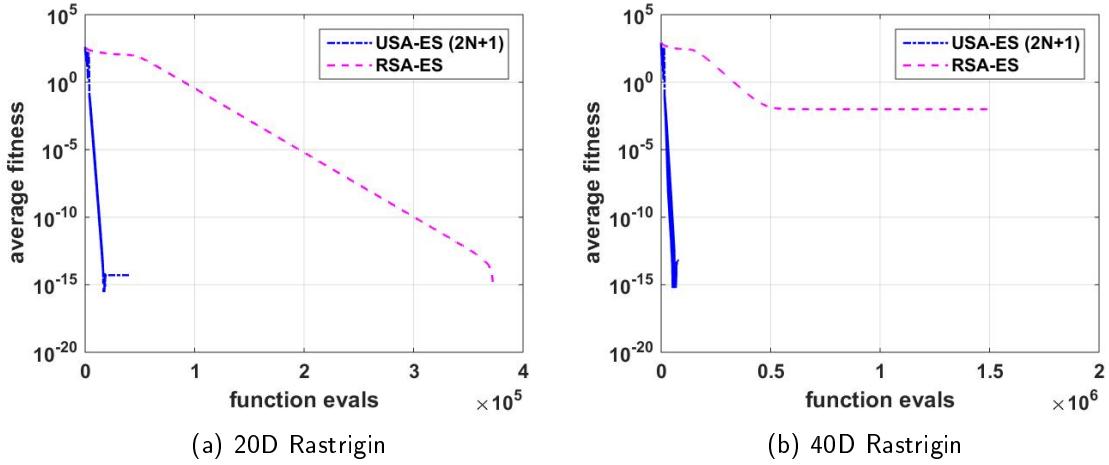


Figure 6.10. Average USA-ES/RSA-ES fitness progression over 100 runs as dimension is increased

6.7 USA-ES Local Optima Analysis

Instead of simply looking at the final result in terms of best fitness found, this next set of trials investigates how close the USA-ES is to finding locally optimal solutions. In order to conduct this experiment, the analytical gradient of each test function from Table 6.1 is determined. Equations (6.6-6.13) are the component-wise analytical derivatives of the benchmark test functions. These derivatives are used to find the L2 norm of the gradient at each point in the algorithm run. This provides information in terms of how often these algorithms are able to escape locally optimal points. In addition, it can help illustrate how close the algorithm is to locally optimal (or globally optimal) points when it converges to a solution.

$$\frac{\partial f_{Ackley}(\bar{x})}{\partial x_i} = \frac{4 \exp\left(-0.2\sqrt{\frac{x_i^2}{n}}\right)\sqrt{\frac{x_i^2}{n}}}{x_i} + \frac{2\pi \sin(2\pi x_i) \exp\left(\frac{\cos(2\pi x_i)}{n}\right)}{n} \quad (6.6)$$

$$\frac{\partial f_{Bohachevsky}(\bar{x})}{\partial x_i} = \begin{cases} 2x_i + 0.9\pi \sin(3\pi x_i) & \text{if } i = 1 \\ 4x_i + 1.6\pi \sin(4\pi x_i) & \text{if } i = n \\ 6x_i + 0.9\pi \sin(3\pi x_i) + 1.6\pi \sin(4\pi x_i) & \text{else} \end{cases} \quad (6.7)$$

$$\frac{\partial f_{Griewank}(\bar{x})}{\partial x_i} = \frac{x_i}{2000} + \frac{\sin\left(\frac{x_i}{\sqrt{i}}\right)}{\sqrt{i}} \prod_{\substack{k=1 \\ k \neq i}}^n \cos\left(\frac{x_k}{\sqrt{k}}\right) \quad (6.8)$$

$$\frac{\partial f_{Rastrigin}(\bar{x})}{\partial x_i} = 2[10\pi \sin(2\pi x_i) + x_i] \quad (6.9)$$

$$\frac{\partial f_{RastScaled}(\bar{x})}{\partial x_i} = 2 \times 10^{\left(\frac{i-1}{n-1}\right)} \left(10\pi \sin\left(2\pi 10^{\left(\frac{i-1}{n-1}\right)} x_i\right) + 10^{\left(\frac{i-1}{n-1}\right)} x_i \right) \quad (6.10)$$

$$\frac{\partial f_{RastSkewed}(\bar{x})}{\partial x_i} = \begin{cases} 200(\pi \sin(20\pi x_i) + x_i) & \text{if } x_i > 0 \\ 2(10\pi \sin(2\pi x_i) + x_i) & \text{else} \end{cases} \quad (6.11)$$

$$\frac{\partial f_{Schaffer}(\bar{x})}{\partial x_i} = \begin{cases} \frac{0.5x_i \left(\sin^2 \left(50(x_{i+1}^2 + x_i^2)^{0.1} \right) + 1 \right)}{(x_{i+1}^2 + x_i^2)^{0.75}} + \frac{10x_i \sin \left(100(x_{i+1}^2 + x_i^2)^{0.1} \right)}{(x_{i+1}^2 + x_i^2)^{0.65}} & \text{if } i = 1 \\ \frac{0.5x_i \left(\sin^2 \left(50(x_i^2 + x_{i-1}^2)^{0.1} \right) + 1 \right)}{(x_i^2 + x_{i-1}^2)^{0.75}} + \frac{10x_i \sin \left(100(x_i^2 + x_{i-1}^2)^{0.1} \right)}{(x_i^2 + x_{i-1}^2)^{0.65}} & \text{if } i = n \\ \frac{0.5x_i \left(\sin^2 \left(50(x_{i+1}^2 + x_i^2)^{0.1} \right) + 1 \right)}{(x_{i+1}^2 + x_i^2)^{0.75}} + \frac{10x_i \sin \left(100(x_{i+1}^2 + x_i^2)^{0.1} \right)}{(x_{i+1}^2 + x_i^2)^{0.65}} \\ + \frac{0.5x_i \left(\sin^2 \left(50(x_i^2 + x_{i-1}^2)^{0.1} \right) + 1 \right)}{(x_i^2 + x_{i-1}^2)^{0.75}} + \frac{10x_i \sin \left(100(x_i^2 + x_{i-1}^2)^{0.1} \right)}{(x_i^2 + x_{i-1}^2)^{0.65}} & \text{else} \end{cases} \quad (6.12)$$

$$\frac{\partial f_{Schwefel}(\bar{x})}{\partial x_i} = -\frac{x_i^2 \cos(\sqrt{|x_i|})}{2|x_i|^{3/2}} - \sin(\sqrt{|x_i|}) \quad (6.13)$$

By looking at Equations (6.6-6.13), it can quickly be determined that many of them approach

zero, as the global optimum is reached. However, the Ackley function along with the Schaffer function require the application of L'Hopital's rule or a numerical analysis to determine what value should be expected as x_i approaches zero. This is potentially the case with the Schwefel function as well, however, the global optimal in this function is actually $(\bar{x}_{Schwefel}^* = [420.96874636]^n)$ where division by zero is avoided. Figure 6.11 illustrates a numerical analysis of what value the gradient of the Ackley and Schaffer functions asymptotically approach as \bar{x} approaches zero to within machine precision. It can be seen that the gradient of the Ackley function should approach four as the global optimum is reached and the gradient of Schaffer approaches infinity. Given that the USA-ES consistently finds the global optimum for both of these problems, it is important to know what values to expect prior to running these experiments.

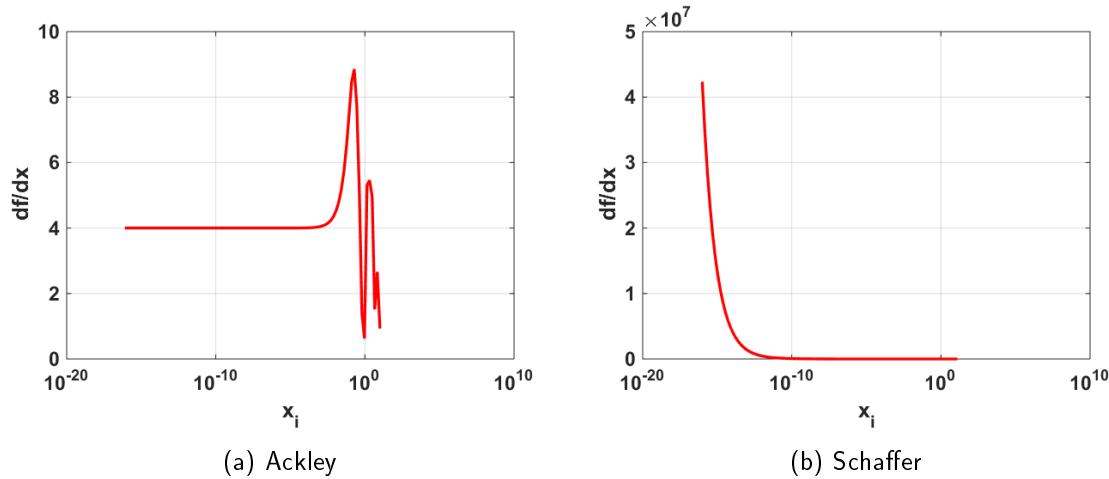


Figure 6.11. Numerical gradient limit tests

With this known, an experiment is conducted where the USA-ES is run using the same settings as before where the algorithm is allowed to run for 1,000 generations and is applied to each problem 100 times so that an average result can be obtained. The L2 norm of the gradient is determined for each run and the average found for each problem is recorded in Table 6.9. It can be seen that in all cases, the average of all USA-ES solutions either has a gradient value that is approximately zero, or has a gradient value that reflects the global optimal solution where the gradient is undefined. This analysis is important because it shows that, in the cases where the USA-ES fails to find the global optimum for all runs (e.g.

Schwefel) it still succeeds in finding a locally optimal solution. Further analysis in Table 6.9 illustrates that this does not change as the dimension is increased up to 40-D.

Function	10-D $\ \nabla f(\bar{x})\ _{avg}$	20-D $\ \nabla f(\bar{x})\ _{avg}$	40-D $\ \nabla f(\bar{x})\ _{avg}$
Ackley	4.0	4.0	4.0
Bohachevsky	1.4×10^{-7}	6.8×10^{-7}	9.9×10^{-7}
Griewank	9.6×10^{-9}	2.0×10^{-8}	2.1×10^{-8}
Rastrigin	2.2×10^{-6}	1.2×10^{-5}	2.4×10^{-5}
Scaled Rastrigin	1.4×10^{-5}	3.4×10^{-5}	6.8×10^{-5}
Skewed Rastrigin	5.6×10^{-6}	8.1×10^{-5}	2.1×10^{-4}
Schaffer	NaN	2.6×10^{53}	4.8×10^{27}
Schwefel	2.5×10^{-7}	1.6×10^{-6}	3.4×10^{-6}

Table 6.9. USA-ES average gradient L2 norms of final solutions

Next, the L2 gradient norm values are tracked as the USA-ES evolves toward a solution. Figure 6.12 illustrates what this gradient norm progression looks like for a typical run of the algorithm on several of the benchmark problems. It can be seen that during the initial phase of the algorithm, when the covariance is large, the gradient magnitudes oscillate and even increase in value as the algorithm jumps from one basin to the next. This represents the exploration phase where the algorithm has a greater chance of escaping basins containing local optima. However, it can also be seen that the algorithm settles into a given basin fairly quickly and begins evolving toward that locally (or globally) optimal point as the covariance is decreased over time and the USA-ES transitions from exploration to exploitation. This is all in line with how the algorithm was designed to work by using a covariance “cooling” scheme.

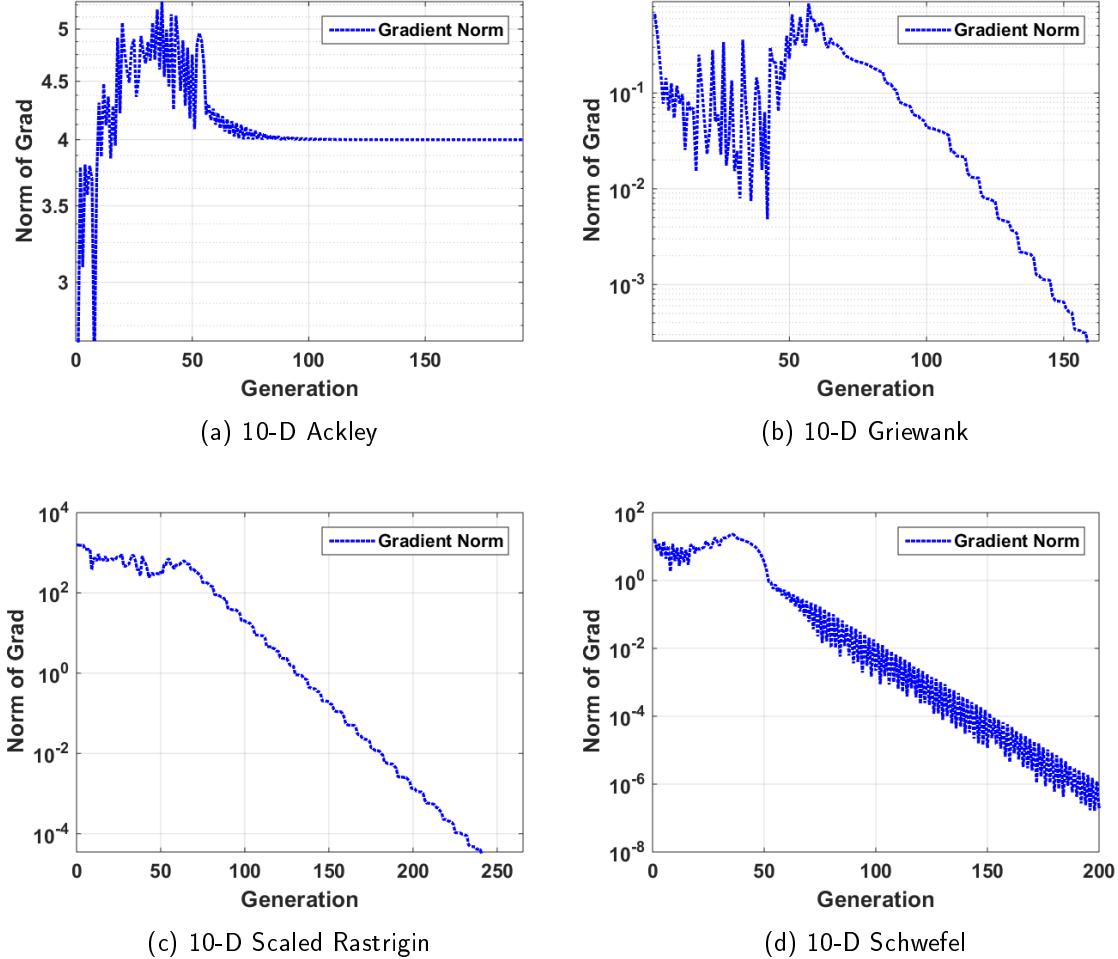


Figure 6.12. USA-ES gradient L2 norm progression on 10-D test functions over a single run

6.8 USA-ES vs. CMA-ES Comparison

The CMA-ES is widely considered to be the state-of-the-art in ESs [189], [191] and is therefore used as a comparison to see how well the USA-ES performs on these multimodal problems relative to some of the more competitive ESs. In these experiments, the open source MATLAB code is utilized with the user-defined settings outlined in Table 6.10. The settings used, are the same ones that Hansen et al. utilized when they applied this algorithm to this same set of benchmark test function [189]. Other parameter settings not stated in

the previously referenced paper are the ones that are utilized in their open source code as summarized in Table 6.10 [187]. It is important to note that this is an educational version of the CMA-ES and a great deal of time has not been expended in optimally tuning the strategy parameters and learning rates for the individual problems. Although, it is worth mentioning that the same is true for the USA-ES regarding the optimal cooling rates. In the following experiments, the average fitness progressions are observed for both algorithms as they are run 100 times with a stopping criteria set to $g_{max} = 10^3$. Again, this appears to be sufficient for both algorithms to stabilize and converge to solutions, as is seen in Figure 6.13.

Parameter	Value
σ_0	$\frac{(\text{Init Range}_{upper} - \text{Init Range}_{lower})}{2}$
C_0	I
$\bar{s}_\sigma^{(0)}$	$[0, \dots, 0]$
$\bar{s}_C^{(0)}$	$[0, \dots, 0]$
μ_ρ	$\frac{1}{\sum_{j=1}^\rho w_j^2}$
c_σ	$\frac{\mu_\rho + 2}{n + \mu_\rho + 5}$
c_C	$\frac{4 + \frac{\mu_\rho}{n}}{n + 4 + \frac{2\mu_\rho}{n}}$
d	$1 + 2\max\left\{0, \sqrt{\frac{\mu_\rho}{n}}\right\} + c_\sigma$
c_1	$\frac{2}{(n+1.3)^2 + \mu_\rho}$
c_ρ	$\frac{2\mu_\rho - 2 + 1/\mu_\rho}{(n+2)^2 + \mu_\rho}$

Table 6.10. CMA-ES user-defined strategy parameter settings

Function	$f(\bar{x})$ Evals	USA $f_{avg\ best}(\bar{x})$ 99% CI	CMA $f_{avg\ best}(\bar{x})$ 99% CI
Ackley	21,000	$1.1 \times 10^{-14} \pm 0.0$	$3.9 \times 10^{-15} \pm 1.0 \times 10^{-16}$
Bohachevsky	21,000	$9.2 \times 10^{-16} \pm 1.2 \times 10^{-17}$	0.0957 ± 0.0186
Griewank	21,000	0.0219 ± 0.0013	$0.0032 \pm 4.6 \times 10^{-4}$
Rastrigin	21,000	$3.4 \times 10^{-14} \pm 9.4 \times 10^{-16}$	6.4473 ± 0.2151
Scaled Rastrigin	21,000	$6.9 \times 10^{-14} \pm 3.0 \times 10^{-16}$	10.7256 ± 0.5568
Skewed Rastrigin	21,000	8.1985 ± 0.1125	14.5463 ± 0.3863
Schaffer	21,000	$3.6 \times 10^{-20} \pm 5.6 \times 10^{-22}$	0.3605 ± 0.0999
Schwefel	21,000	770.0818 ± 26.4045	$1,709.6 \pm 28.2987$

Table 6.11. 10-D USA-ES vs. CMA-ES results ($\rho = 10$ $\lambda = 21$)

Function	$f(\bar{x})$ Evals	USA $f_{avg\ best}(\bar{x})$ 99% CI	CMA $f_{avg\ best}(\bar{x})$ 99% CI
Ackley	41,000	$7.7 \times 10^{-15} \pm 1.7 \times 10^{-16}$	$6.1 \times 10^{-15} \pm 1.4 \times 10^{-16}$
Bohachevsky	41,000	0 ± 0	0.0507 ± 0.0113
Griewank	41,000	0.0234 ± 0.0015	$1.5 \times 10^{-4} \pm 8.5 \times 10^{-5}$
Rastrigin	41,000	$7.4 \times 10^{-15} \pm 1.2 \times 10^{-15}$	10.2182 ± 0.2696
Scaled Rastrigin	41,000	$1.4 \times 10^{-16} \pm 1.2 \times 10^{-16}$	17.8595 ± 0.5970
Skewed Rastrigin	41,000	7.5119 ± 0.1571	24.8839 ± 0.4100
Schaffer	41,000	$0.0012 \pm 4.9 \times 10^{-4}$	0.2427 ± 0.0680
Schwefel	41,000	631.8764 ± 8.7571	3973.4 ± 42.0239

Table 6.12. 20-D USA-ES vs. CMA-ES results ($\rho = 20 \lambda = 41$)

Function	$f(\bar{x})$ Evals	USA $f_{avg\ best}(\bar{x})$ 99% CI	CMA $f_{avg\ best}(\bar{x})$ 99% CI
Ackley	81,000	$4.1 \times 10^{-10} \pm 7.7 \times 10^{-12}$	$7.5 \times 10^{-15} \pm 1.0 \times 10^{-16}$
Bohachevsky	81,000	0 ± 0	0.0482 ± 0.0128
Griewank	81,000	0.0213 ± 0.0019	0 ± 0
Rastrigin	81,000	$6.0 \times 10^{-14} \pm 2.3 \times 10^{-15}$	15.0040 ± 0.2620
Scaled Rastrigin	81,000	$8.9 \times 10^{-15} \pm 1.1 \times 10^{-15}$	30.8835 ± 0.7558
Skewed Rastrigin	81,000	3.1142 ± 0.1043	44.7134 ± 0.3760
Schaffer	81,000	0.0069 ± 0.0012	0.0907 ± 0.0352
Schwefel	81,000	$1,305.0 \pm 1.7716$	$8,190.7 \pm 69.5001$

Table 6.13. 40-D USA-ES vs. CMA-ES results ($\rho = 40 \lambda = 81$)

Tables 6.11-6.13 illustrate that the USA-ES matches, or outperforms the CMA-ES on the majority of the problems, with the exception of the Griewank function, when using the same number of offspring, $\lambda = 2n + 1$ and recombination fraction $\rho = \lambda / 2$. Figure 6.13 gives a more detailed look at the average fitness progression for several of the 40-D benchmark functions. It can be seen that the CMA-ES outperforms the USA-ES on the Griewank problem, while the USA-ES offers superior performance on the Bohachevsky and various Rastrigin problems. Again, it is important to emphasize that this experiment is evaluating the average fitness and average best solution found over 100 runs. Even though, both algorithms may be finding globally optimal solutions on the many of the runs, the average can be negatively impacted by a small number of runs where the algorithm converges to sub-optimal points. However, this experimental approach gives the reader a good understanding of how consistently each algorithm is able to locate optimal solutions, or more specifically solutions that have the best fitness values.

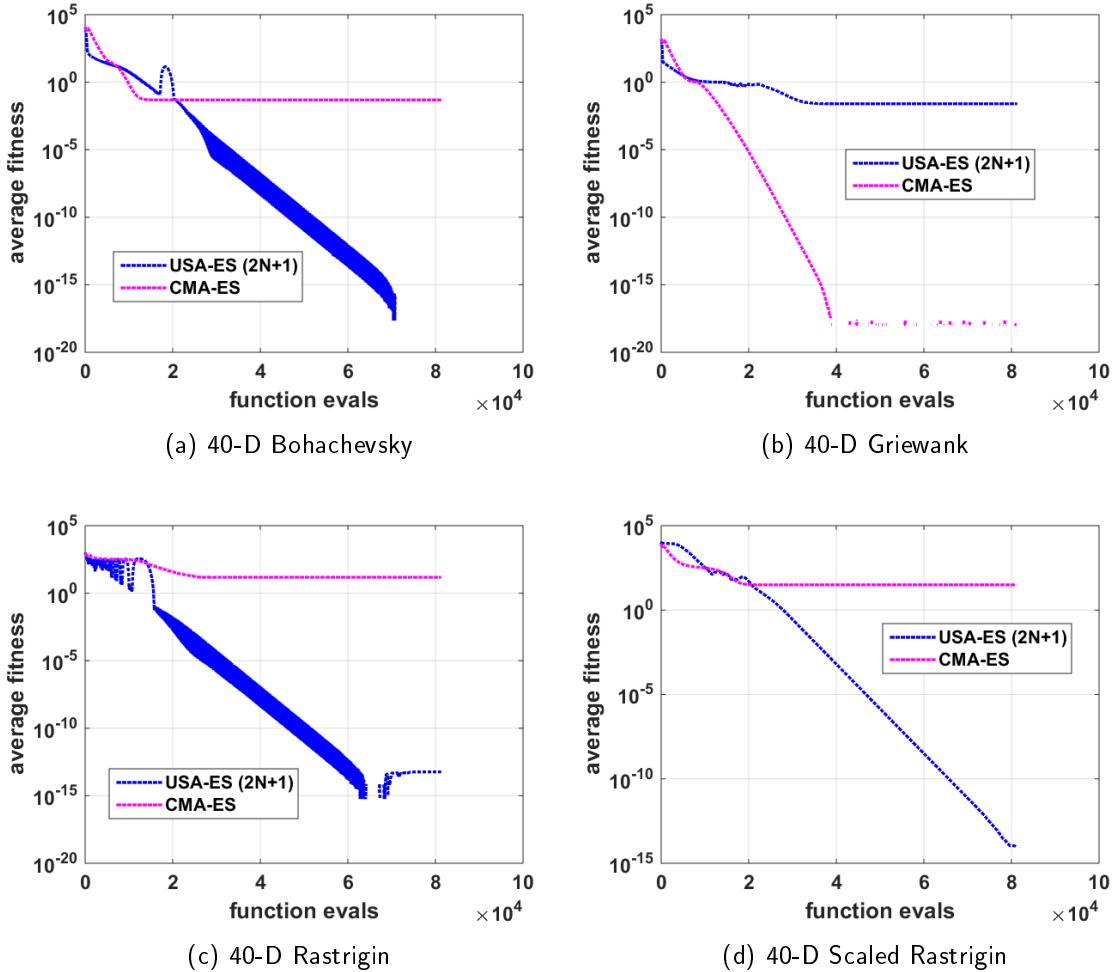


Figure 6.13. Average USA-ES/CMA-ES fitness progression on 40-D test functions with equal number of sampling points

Table 6.14 illustrates the percentage of runs that result in a solution that produced a fitness value less than 10^{-8} . This indicates how often each algorithm was able to obtain a globally optimal solution during the 100 trials. It can be seen that the results illustrate similar trends as were seen above in Tables 6.11-6.13.

Function	USA % Success	CMA % Success
Ackley 10-D	100	100
Ackley 20-D	100	100
Ackley 40-D	100	100
Bohachevsky 10-D	100	82
Bohachevsky 20-D	100	88
Bohachevsky 40-D	100	90
Griewank 10-D	10	70
Griewank 20-D	12	98
Griewank 40-D	17	100
Rastrigin 10-D	100	0
Rastrigin 20-D	100	0
Rastrigin 40-D	100	0
Scaled Rastrigin 10-D	100	0
Scaled Rastrigin 20-D	100	0
Scaled Rastrigin 40-D	100	0
Skewed Rastrigin 10-D	0	0
Skewed Rastrigin 20-D	0	0
Skewed Rastrigin 40-D	1	0
Schaffer 10-D	100	15
Schaffer 20-D	96	13
Schaffer 40-D	0	7
Schwefel 10-D	3	0
Schwefel 20-D	0	0
Schwefel 40-D	0	0

Table 6.14. USA-ES vs. CMA-ES success probabilities

6.8.1 USA-ES/CMA-ES Solution Quality Study

The following set of experiments is identical to the ones conducted in Section 6.6.1, with the exception that the CMA-ES is compared with the USA-ES. This experiment increases the number of sample points used within the CMA-ES until it achieves a final average fitness that is comparable to that of the USA-ES on the problems where the USA-ES achieves a better average fitness than the CMA-ES when using the same number of points. More specifically, the number of points are incremented by ten and the CMA-ES is run 100 times for each case so that a new average fitness evolution can be determined. If this new average fitness progression achieves a final value that is within 10^{-6} of the fitness found by the USA-ES then this number of points and respective fitness value is stored. If it is not within

the given tolerance, the number of points is increased and the process continues. In order to avoid infinite loops, a limit on the upper bound of sampling points has been set to 1,501. Tables 6.15-6.17 summarize these results.

Function	USA-ES $f(\bar{x})$ Evals	CMA-ES $f(\bar{x})$ Evals	Runtime $\frac{CMA}{USA}$
Ackley	21,000	21,000	1.61
Bohachevsky	21,000	61,000	4.55
Griewank	21,000	21,000	1.62
Rastrigin	21,000	641,000	46.10
Scaled Rastrigin	21,000	681,000	43.53
Skewed Rastrigin	21,000	1,501,000 (limit)	161.92
Schaffer	21,000	111,000	7.22
Schwefel	21,000	131,000	7.53

Table 6.15. USA-ES/CMA-ES 10D solution quality results

Function	USA-ES $f(\bar{x})$ Evals	CMA-ES $f(\bar{x})$ Evals	Runtime $\frac{CMA}{USA}$
Ackley	41,000	41,000	1.49
Bohachevsky	41,000	101,000	3.21
Griewank	41,000	41,000	1.44
Rastrigin	41,000	891,000	30.52
Scaled Rastrigin	41,000	921,000	19.67
Skewed Rastrigin	41,000	1,501,000 (limit)	50.33
Schaffer	41,000	191,000	4.04
Schwefel	41,000	1,501,000 (limit)	29.55

Table 6.16. USA-ES/CMA-ES 20D solution quality results

Function	USA-ES $f(\bar{x})$ Evals	CMA-ES $f(\bar{x})$ Evals	Runtime $\frac{CMA}{USA}$
Ackley	81,000	81,000	1.47
Bohachevsky	81,000	91,000	3.33
Griewank	81,000	81,000	1.48
Rastrigin	81,000	1,371,000	30.39
Scaled Rastrigin	81,000	1,421,000	19.85
Skewed Rastrigin	81,000	1,501,000 (limit)	48.64
Schaffer	81,000	221,000	3.94
Schwefel	81,000	1,501,000 (limit)	29.37

Table 6.17. USA-ES/CMA-ES 40D solution quality results

It can be seen that the CMA-ES competes well with the USA-ES when more sample points are used. This result supports the findings by Hansen et al. in their study on applying the CMA-ES to this same set of multimodal functions [189]. In fact, the CMA-ES is able to get as good, if not better, average fitness values on all but the Skewed Rastrigin and Schwefel functions in the 20-D and 40-D cases. In the 10-D case, the CMA-ES is able to get to solutions with average fitness values that are as good or better than the USA-ES in all but the Skewed Rastrigin problem. This study does illustrate that the CMA-ES is negatively impacted with increasing dimension when compared to the USA-ES, but this does not have as big of an impact on the CMA-ES as it did with regard to the RSA-ES.

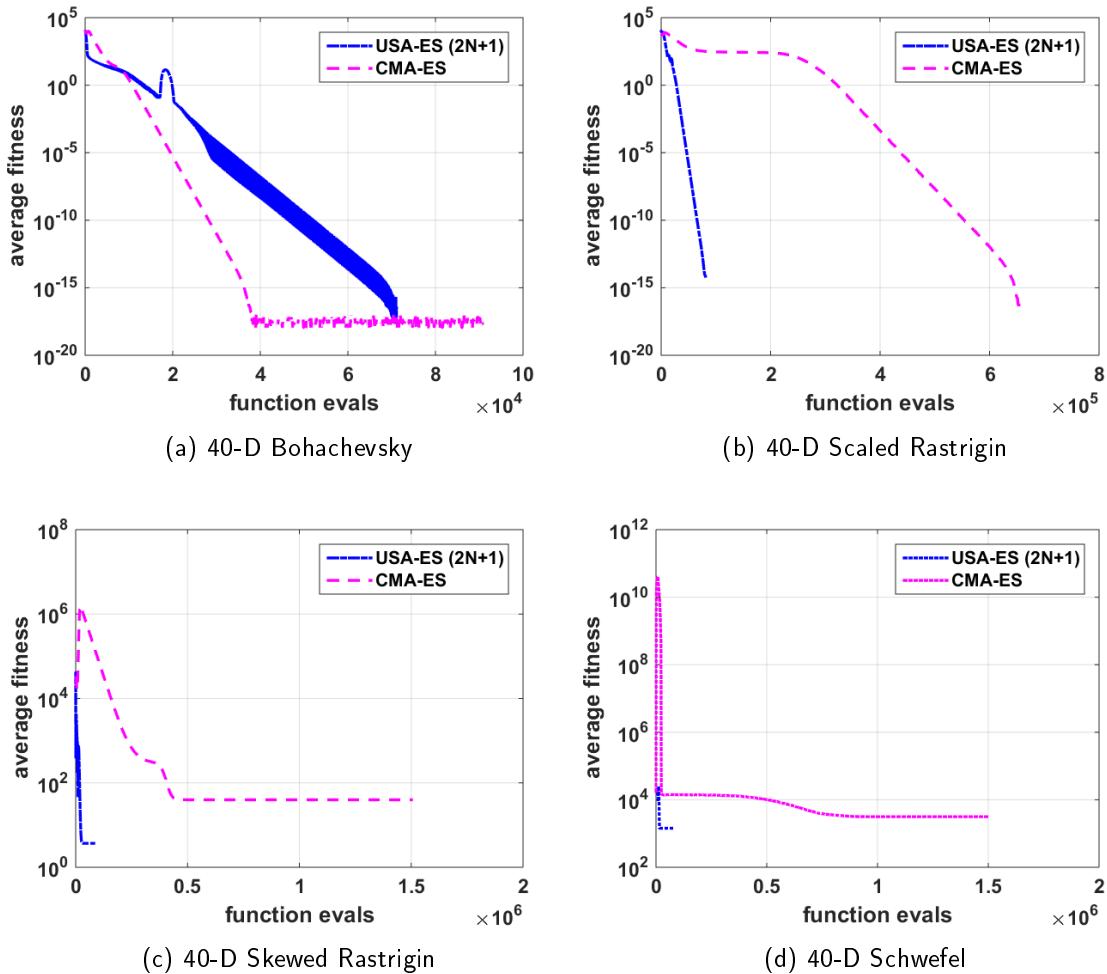


Figure 6.14. Average USA-ES/CMA-ES fitness progression on 40-D test functions with unequal numbers of sampling points

Figure 6.14 illustrates the average fitness progressions for several of these functions. It can be seen that the CMA-ES only required ten additional points in the 40-D Bohachevsky case to match or exceed the USA-ES performance. In the 40-D Scaled Rastrigin problem, the CMA-ES required 1,421 sample points to match the solution quality of the USA-ES which was using 81 sample points. Lastly, it can be seen that the 40-D Skewed Rastrigin and Schwefel functions proved to be the most challenging for the CMA-ES in terms of matching solution quality with that of the USA-ES. However, it must be noted that these two problems are also difficult for the USA-ES in terms of consistently finding the global minimum. It is worth pointing out that the USA-ES is unable to achieve an optimal covariance adaptation rate given the simple exponential cooling schedule used. This indicates that a lot of the performance gain that is observed in these experiments results directly from the use of deterministic sampling.

6.9 GHSA-ES

Given that some of the benchmark test problems have proven to be difficult, even for the USA-ES, the application of a different Gaussian distribution parameterization is investigated so that the number of deterministically chosen points and associated order can be increased. While it is possible to increase the order of the moment-matching sigma points, it is a fairly involved process [199]. For this reason, Gauss-Hermite points are utilized to parametrize the n-dimensional normal distributions used for mutation within these algorithms. Probabilistic Gauss-Hermite points are selected, as opposed to other interpolation points such as standard Gauss, given that they are well-suited to parameterize normal distributions over an infinite range, $[-\infty, \infty]$. [200] With this, the GHSA-ES will be the same as the USA-ES in every way, except for the parametrization nodes used. Instead of using 3rd order sigma points, it uses a higher order set of points so that it might be able to gather more information about the n-dimensional fitness landscape during each generation of the algorithm. This section investigates the trades of using Gauss-Hermite points as a parametrization technique when compared to the USA-ES with 3rd order sigma points.

6.9.1 Gauss-Hermite Parameterization Using Smolyak Sparse Grids

It is quickly determined that a sparse grid representation is needed to somewhat curb the curse of dimensionality. When using a standard interpolation point mesh, the number of

interpolation points grows exponentially with regard to dimension, $\mathcal{O}(\text{points}^n)$, due to the use of tensor products to create the n-dimensional grid [200]. For this reason, Smolyak sparse grids are implemented in the GHSA-ES so that this curse of dimensionality is somewhat manageable for lower accuracy levels and lower dimensional search spaces [198]. Figure 6.15 illustrates how the number of Gauss-Hermite sampling points grows with regard to dimension and accuracy level when using Smolyak sparse grids. It must be noted that the order, \mathcal{O} , of these interpolation points is equal to: $\mathcal{O} = 2L - 1$ where L is the accuracy level. An open source MATLAB function (“nwspgr.m”) written by Florian Heiss and Viktor Winschel is employed to generate these sets of interpolation points on sparse grids.

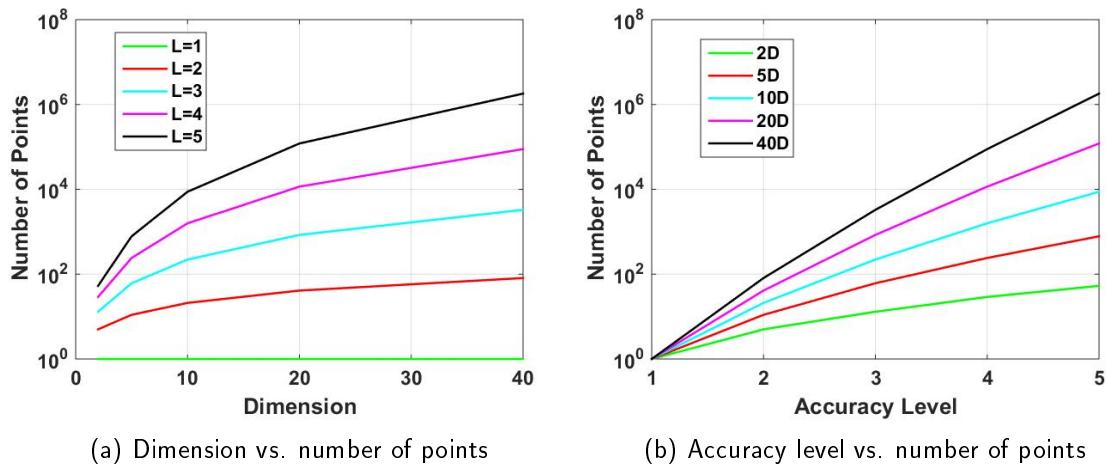


Figure 6.15. Gauss-Hermite point growth w.r.t. accuracy level and dimension using Smolyak sparse grids

Figure 6.16 illustrates the increased number of points that can be obtained by going from 3rd order sigma points to higher order Gauss-Hermite points using sparse grids. It is discovered that 3rd order sigma points are an equivalent to 3rd order Gauss-Hermite points ($L = 2$) on a Smolyak sparse grid when $\kappa = 1 - n$. This is true for both nodes and weights associated with both sets of points and can be seen in Figure 6.16a.

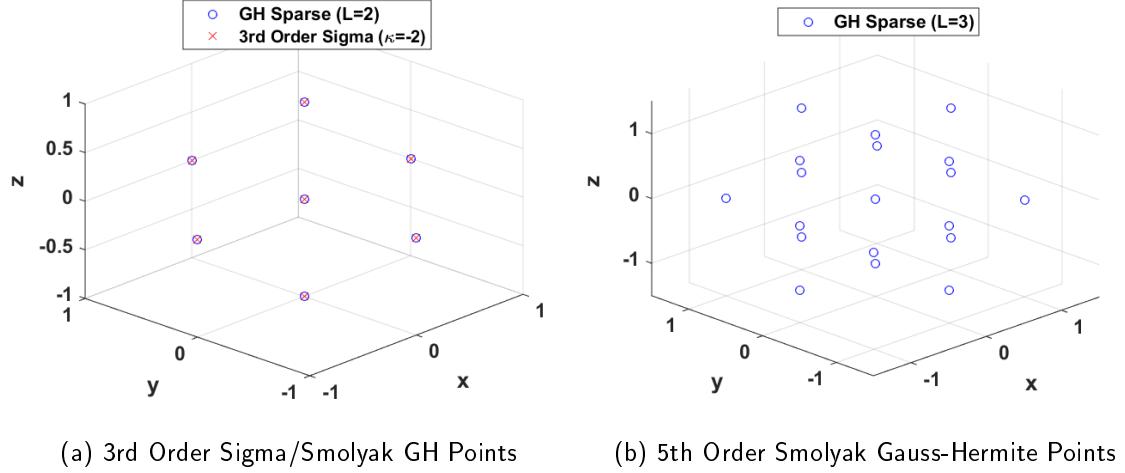


Figure 6.16. Comparison of parameterization methods

6.9.2 USA-ES vs. GHSA-ES

A similar set of experiments is performed on the benchmark test functions to compare the performance of the USA-ES and GHSA-ES. In these experiments, 5th order Gauss-Hermite points are used on a Smolyak sparse grid ($L = 3$) for the GHSA-ES and 3rd order sigma points are used with $\kappa = 1$ on the USA-ES. With this, it must be noted that the two different algorithms are utilizing different numbers of fitness evaluations to arrive at the solution, even though they both have a stopping criteria of one thousand generations, $g_{max} = 10^3$. Furthermore, a ratio of the average computational time required to run the GHSA-ES vs. the USA-ES on each problem is utilized to further investigate the cost associated with using the increased number of points. Tables 6.18-6.20 illustrate the results of these experiments.

Function	USA $f_{avg\ best}(\bar{x})$ 99% CI	GHSA $f_{avg\ best}(\bar{x})$ 99% CI	CPU Time $\frac{GHSA}{USA}$
Ackley	$1.1 \times 10^{-14} \pm 0.0$	$5.9 \times 10^{-15} \pm 1.9 \times 10^{-16}$	9.27
Bohachevsky	$9.2 \times 10^{-16} \pm 1.2 \times 10^{-17}$	$1.6 \times 10^{-16} \pm 8.9 \times 10^{-18}$	9.36
Griewank	0.0219 ± 0.0013	$0.0084 \pm 7.8 \times 10^{-4}$	9.27
Rastrigin	$3.4 \times 10^{-14} \pm 9.4 \times 10^{-16}$	$1.4 \times 10^{-14} \pm 0$	9.34
Scaled Rastrigin	$6.9 \times 10^{-14} \pm 3.0 \times 10^{-16}$	0.0995 ± 0.0245	9.34
Skewed Rastrigin	8.1985 ± 0.1125	7.0145 ± 0.1078	9.18
Schaffer	$3.6 \times 10^{-20} \pm 5.6 \times 10^{-22}$	$1.0 \times 10^{-20} \pm 1.4 \times 10^{-22}$	9.29
Schwefel	770.0818 ± 26.4045	251.8875 ± 14.3952	9.43

Table 6.18. 10-D USA-ES ($\lambda_{USA} = 21$) vs. GHSA-ES ($\lambda_{GHSA} = 221$)

Function	USA $f_{avg\ best}(\bar{x})$ 99% CI	GHSA $f_{avg\ best}(\bar{x})$ 99% CI	CPU Time $\frac{GHSA}{USA}$
Ackley	$7.7 \times 10^{-15} \pm 1.7 \times 10^{-16}$	$7.1 \times 10^{-15} \pm 0$	18.11
Bohachevsky	0 ± 0	0 ± 0	18.51
Griewank	0.0234 ± 0.0015	$0.0071 \pm 5.5 \times 10^{-4}$	17.63
Rastrigin	$7.4 \times 10^{-15} \pm 1.2 \times 10^{-15}$	$5.7 \times 10^{-16} \pm 3.3 \times 10^{-16}$	17.86
Scaled Rastrigin	$1.4 \times 10^{-16} \pm 1.2 \times 10^{-16}$	0.2587 ± 0.0358	18.58
Skewed Rastrigin	7.5119 ± 0.1571	0.2089 ± 0.0388	17.51
Schaffer	$3.6 \times 10^{-20} \pm 5.6 \times 10^{-22}$	$1.2 \times 10^{-10} \pm 1.4 \times 10^{-12}$	17.88
Schwefel	631.8764 ± 8.7571	3.5532 ± 1.6567	18.96

Table 6.19. 20-D USA-ES ($\lambda_{USA} = 41$) vs. GHSA-ES ($\lambda_{GHSA} = 841$)

Function	USA $f_{avg\ best}(\bar{x})$ 99% CI	GHSA $f_{avg\ best}(\bar{x})$ 99% CI	CPU Time $\frac{GHSA}{USA}$
Ackley	$4.1 \times 10^{-10} \pm 7.7 \times 10^{-12}$	$4.0 \times 10^{-11} \pm 3.3 \times 10^{-12}$	36.57
Bohachevsky	0 ± 0	0 ± 0	37.10
Griewank	0.0213 ± 0.0019	$0.0149 \pm 8.3 \times 10^{-4}$	35.78
Rastrigin	$6.0 \times 10^{-14} \pm 2.3 \times 10^{-15}$	$1.3 \times 10^{-14} \pm 1.9 \times 10^{-15}$	36.34
Scaled Rastrigin	$8.9 \times 10^{-15} \pm 1.1 \times 10^{-15}$	0.4477 ± 0.0495	37.19
Skewed Rastrigin	3.1142 ± 0.1043	0 ± 0	35.67
Schaffer	0.0069 ± 0.0012	$2.9 \times 10^{-5} \pm 3.1 \times 10^{-7}$	38.55
Schwefel	$1,305.0 \pm 1.7716$	$2.5 \times 10^{-11} \pm 2.3 \times 10^{-13}$	37.42

Table 6.20. 40-D USA-ES ($\lambda_{USA} = 81$) vs. GHSA-ES ($\lambda_{GHSA} = 3281$)

From these results, it can be seen that the increase in order and number of deterministic sampling points can prove to be beneficial on some of the more challenging problems. However, the performance actually gets worse in terms of accuracy on the Scaled Rastrigin problem in all dimension cases. The greatest benefit from using the 5th order sparse Gauss-Hermite points can be seen on the Skewed Rastrigin and Schwefel functions, especially in the 40-D case where the GHSA-ES is able to locate the global optimal solution on every run. Looking at the previous experiments, this is the first algorithm that is able to do this on these two problems in the 40-D case. However, this requires roughly thirty-six times the computational time over the USA-ES in the 40-D case. This can translate to significantly longer computation times as the expense of the function evaluations increase and as the problem dimension is increased. Furthermore, as the search domain dimension is increased, the number of sparse Gauss-Hermite points can become very large.

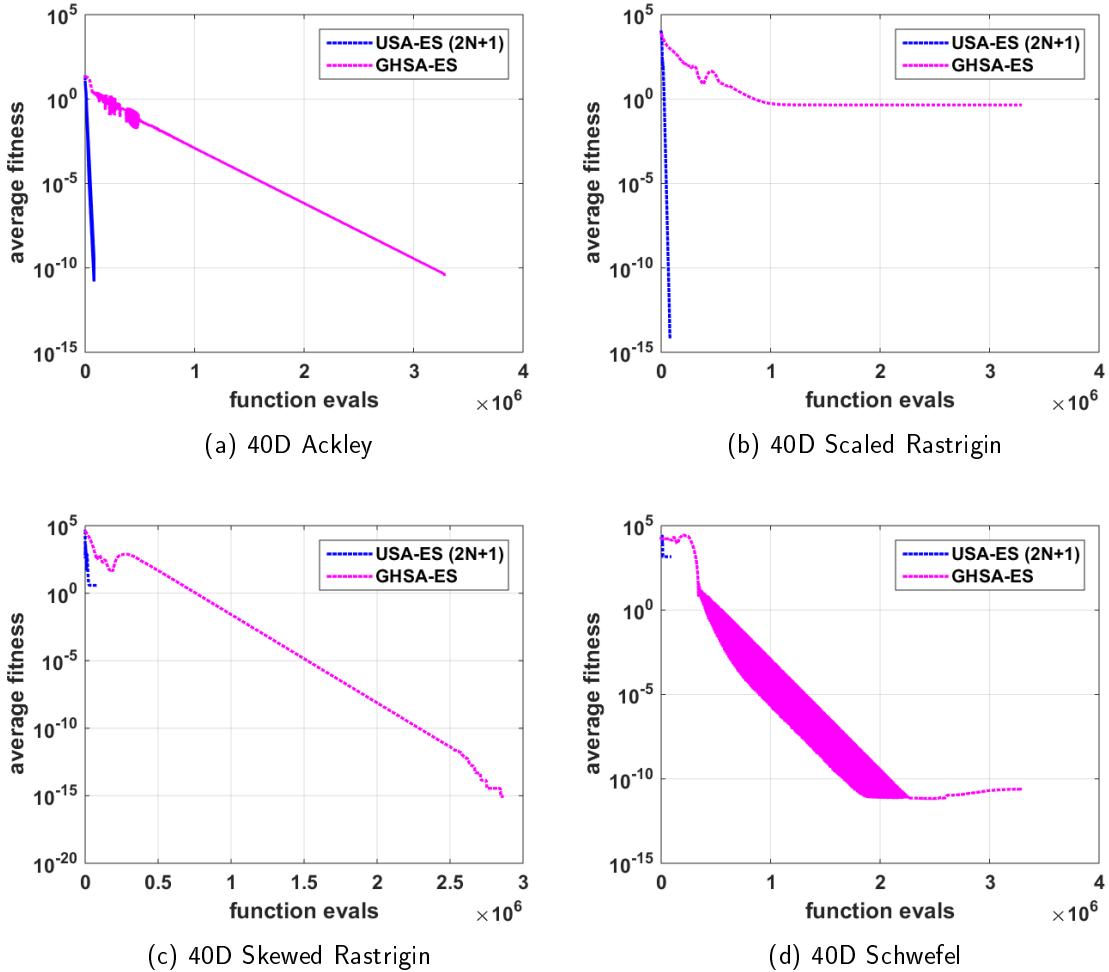


Figure 6.17. USA-ES vs. GHSA-ES on 40D test functions

Figure 6.17 illustrates the average fitness progressions for 100 runs of both ES types on several of the more interesting cases for the 40-D problem set. It can be seen in Figure 6.17a, that the larger number of points slows down the average convergence rate per function evaluation in the fitness domain, even though it is able to obtain better, or similar final solutions on most of the problems in terms of average fitness value. Figure 6.17b illustrates the Scaled Rastrigin function, where the increase in the number of points actually results in less accurate final solutions, as they slow down the progression of the algorithm too much for this relatively “shallow” fitness landscape. Lastly, Figures 6.17c and d illustrate the two problems where the use of higher order points significantly improve the solution

accuracy. These results indicate that the use of the GHSA-ES with higher order points on problems with moderate dimensionality can be a very useful tool in some scenarios where the USA-ES may be having difficulty due to the sparseness of 3rd order sigma points. However, it should be noted that the 3rd order sigma points do surprisingly well given the relatively small number of points used, even in higher dimensional problems.

Table 6.21 illustrates the percentage of runs that result in a solution that produced a fitness value less than 10^{-8} . This indicates how often each algorithm was able to obtain a globally optimal solution during the 100 trials. It can be seen that the results illustrate similar trends as were seen above in Tables 6.18-6.20.

Function	USA % Success	GHSA % Success
Ackley 10-D	100	100
Ackley 20-D	100	100
Ackley 40-D	100	100
Bohachevsky 10-D	100	100
Bohachevsky 20-D	100	100
Bohachevsky 40-D	100	100
Griewank 10-D	10	29
Griewank 20-D	12	45
Griewank 40-D	17	22
Rastrigin 10-D	100	100
Rastrigin 20-D	100	100
Rastrigin 40-D	100	100
Scaled Rastrigin 10-D	100	90
Scaled Rastrigin 20-D	100	74
Scaled Rastrigin 40-D	100	61
Skewed Rastrigin 10-D	0	0
Skewed Rastrigin 20-D	0	82
Skewed Rastrigin 40-D	1	100
Schaffer 10-D	100	100
Schaffer 20-D	96	100
Schaffer 40-D	0	0
Schwefel 10-D	3	10
Schwefel 20-D	0	97
Schwefel 40-D	0	100

Table 6.21. USA-ES vs. GHSA-ES success probabilities

6.10 Discussion and Related Studies

The fundamental background and motivation behind using deterministic sampling in the context of an ES was to increase the algorithm efficiency by choosing a set of meaningful points that could better sample the search space with a minimal number of points. Deterministic sampling is not a new idea, where numerical interpolation, integration, and differentiation methods often rely on making intelligent choices for abscissas in a way that minimize interpolation error [200]. However, the integration and employment of this technique within the context of ESs and GAs is new and unique. The utilization of unscented sigma points is a logical fit when sampling from distributions given their nature and ability to contain a lot of information about the distribution with the minimum number of points. More specifically, the Gaussian 3rd order sigma points represent the minimum number of sampling points for an n-dimensional Gaussian distribution that can exactly match up to 3rd order statistical moments [197]. The logic behind utilizing these points, is that they are more meaningful and have the potential to more efficiently sample the fitness landscape than the conventional method of pure random sampling. They also only increase linearly in number with respect to the problem dimension, which is very important when dealing with high dimensional search spaces. It must be noted that other types of sampling distributions could be considered such as Cauchy, Uniform, Beta, etc. that may offer superior performance on some problems. The Gaussian distribution is selected for this study due to it's popularity within the ES community. In addition, a Gaussian distribution best emulates the process of mutation in Biology where small mutations from the mean are far more likely than large mutations.

In addition to the increased statistical meaning behind these deterministically chosen sampling points, they produce a much more symmetric sampling spread that would require a significant amount of random sampling to replicate. As an analogy, it is similar to fishing with a more evenly spaced fishing net compared to fishing with a net that has a random mesh size with areas that posses very fine meshes and other areas that have very coarse mesh sizes. Without any prior knowledge of the problem space (or where the fish are located) this is not likely the best approach. As the number of random points increases, the mean distances between nodes in the randomly spaced fishing net will become smaller and smaller. The solution quality experiments in Sections 6.6.1 and 6.8.1 along with their respective results reflect this idea, where as the number of sample points is increased, the performance

of the algorithms improves and approaches that of the USA-ES which uses deterministic sampling. It should be noted that the deterministic ESs developed and investigated in this chapter only utilize half of the information associated with the abscissas. Both unscented sigma points and Gauss-Hermite points have weights that are associated with each point that these algorithms currently ignore. In doing this, potentially useful information is discarded by not incorporating these weights into the search updates. In addition, these points are only utilized to update the sampling distribution mean and not the covariance. In fact several authors have shown optimal convergence rates based on simplified ESs operating on simple cost functions such as the spherical function [180]. Knowing these rates, the simplified, simulated annealing style cooling schedule is likely not the optimal way to update the covariance as the search progresses. These are the issues that will ultimately be addressed in Chapter 7 where the nodes and associated weights are fully employed to intelligently adapt both the sampling distribution mean and covariance.

The literature research conducted indicates that the use of entirely deterministic sampling within the context of an ES that results in a repeatable algorithm is a new contribution. However, there have been several authors that have been investigating ideas where the randomness of the sampling is reduced through the process of reflecting, or mirroring, sample points across the search space. Auger et al. conducted a study where they investigate the idea of mirroring a portion of the randomly sampled points [201]. They investigate three different mirroring schemes: one where the points to be mirrored are randomly selected, another where a lowest performing subset of the sample points are mirrored, and the last one where the lower performing subset are mirrored based on direction, but the magnitude of the mirrored vector are randomly rescaled. In these schemes, after the points are mirrored, the better of the two (un-mirrored vs. mirrored) is selected to become part of the offspring population. In developing these techniques, they were able to theoretically and empirically prove increases in convergence rates of up to 56% over traditional ESs with recombination on a spherical cost function [201]. Furthermore, they conclude that this benefits ESs with relatively small populations significantly more than algorithms with large populations given that larger populations are likely to already have points close to where the mirrored points might lie [201]. While their methods of mirroring points are fundamentally different than using sets of predetermined abscissas for sampling, this finding supports the fishing net analogy idea where larger numbers of random points begin to generate increasingly

symmetric sample sets, thus decreasing the advantage that deterministic methods have over Monte Carlo schemes. They also point out that mirroring all of the points can result in a bias toward small update steps which has the potential to lead algorithms to prematurely converge. This is the reason why they implement a down select between mirrored points and introduce a method that randomly scales the mirrored point vector magnitude [201]. This bias is avoided in both the USA-ES and GHSA-ES due to the carefully controlled covariance adaptation scheme. However, this is something that needs to be kept in mind when these points are fully leveraged in both the mean and covariance updates, as will be seen in Chapter 7.

6.11 Application of USA-ES To Trajectory Optimization

The final section of this chapter focuses on an application of the USA-ES algorithm to solve the discretized lunar lander optimal control problem (Equation (4.3)) introduced in Section 4.2. This is a step toward more realistic optimal control problems that will put the USA-ES to the test in terms of dimensionality and real-world applicability. Before this algorithm is directly applied to a lunar lander optimal control problem, several modifications and improvements are implemented to help enhance the global search capability and computational speed of the USA-ES.

6.11.1 Further Enhancing The USA-ES For Global Search

The USA-ES algorithm, as described in Section 6.3 above, appears to have a set number of search points as a function of the problem dimension. With sigma points, it can be fairly difficult to generate higher order sets of points. It has been shown that other parameterizations can be utilized, such as Gauss-Hermite points on sparse grids. However, these methods still suffer from the curse of dimensionality, making them difficult to apply on the high-dimensional search domains that often result from time discretization. If a standard ES with random sampling is not providing sufficient solutions, the user can always increase the number of sample points. Unfortunately, it is not this straightforward when using parameterizations of n-dimensional mutation distributions. Fortunately there are several techniques that are not being utilized in the above simulations that can lead to improved performance of the USA-ES on multimodal functions without changing the

number of sampling points. These ideas are employed on the USA-ES for the trajectory optimization problem and described in the following sections of this chapter.

6.11.2 Covariance Resets

One idea that has been alluded to earlier is the idea of resetting the covariance to its initial values allowing the algorithm to continue to explore the search space after it has initially converged to a local solution. The convergence can be measured by looking at the average fitness value or the difference between parent values from one generation to the next to determine if it has stagnated or settled at a certain value. This technique is added to the USA-ES algorithm so that it is able to continue searching different areas by transitioning from an increasingly local distribution (small covariance) to a global one (large covariance). This resets the covariance to the initial values every time the average fitness from generation to generation varies less than a user defined tolerance. In the case of the below simulations, the tolerance is set to 10^{-6} .

6.11.3 Parallel Implementation of The USA-ES

In addition to the covariance reset concept, the USA-ES is no different from other evolutionary algorithms in that it lends itself very nicely to parallel computing. One way to take advantage of this is to use a parallel island architecture where multiple processors run the same USA-ES starting at different initial points in the search space. The islands (or processors) then share data or have migrations every so many generations so that they may help the other island populations leverage their successful searches. This idea again stems from biology with the concept of isolated species at different geographical locations that migrate every so often to different areas affecting those regional populations. In order to implement this in a more efficient way on the USA-ES, the algorithm is coded in C versus MATLAB so that it can run faster using a lower-level language. This is especially useful when large numbers of iterations are desired ($g_f = 5 \times 10^6$ in this case). Message passing interface (MPI) is then utilized to implement a parallel island version of the algorithm so that it can be run on multiple processors simultaneously. The stopping criteria is simply a set number of generations and the user defines a migration frequency in which the processors share their best solution found to date with a neighboring processor using a simple round-robin network architecture illustrated in Figure 6.18. This parallel implementation is

done by inserting or “migrating” the best solution from one island population into the sorted set of offspring from the receiving population in a way that it replaces the best offspring member that it beats in terms of fitness. It must be noted that this best solution is able to pull the various USA-ES algorithms that are running in parallel toward better regions of the search space. This effect happens within the recombination operator, where the newly migrated solution vector is factored into the creation of a new parent vector or distribution mean. The degree to which this migration affects the search distribution of the receiving processor is dependent on the relative differences in fitness value and Euclidean distance between the migrated solution vector and the rest of the ρ offspring vectors being utilized in the recombination operator. In the case where the relative fitness values are very similar, there may be an instance where the migrated solution vectors are unable to pull given search distributions out of locally optimal basins. However, it must be noted that in this scenario, the parallelization of the algorithm still enhances the algorithm’s ability for global search through the different starting points for each processor.

There are a number of different architectures and migrations schemes where several solutions are shared at a time. However, for the purposes of this study, the simple round-robin architecture with single solution migrations will be used. This parallel construct allows for faster convergence and improved search capability by helping each algorithm explore areas that seem to be paying off the most in terms of good objective function values. A number of studies have been conducted regarding these types of parallel island architectures and what attributes lead to efficient algorithms [43], [202]. Unfortunately, the theoretical analysis is somewhat lacking and is closely tied to problem-dependent characteristics that are not easily known *a priori* so empirical analysis and tuning is often required to optimize these parallel algorithms [202], [203]. For the purposes of this chapter, some empirical analysis has influenced the choices for parameters such as the number of processors and migration frequency used in the following sections, but it is not discussed here in detail nor is it considered to be optimal.

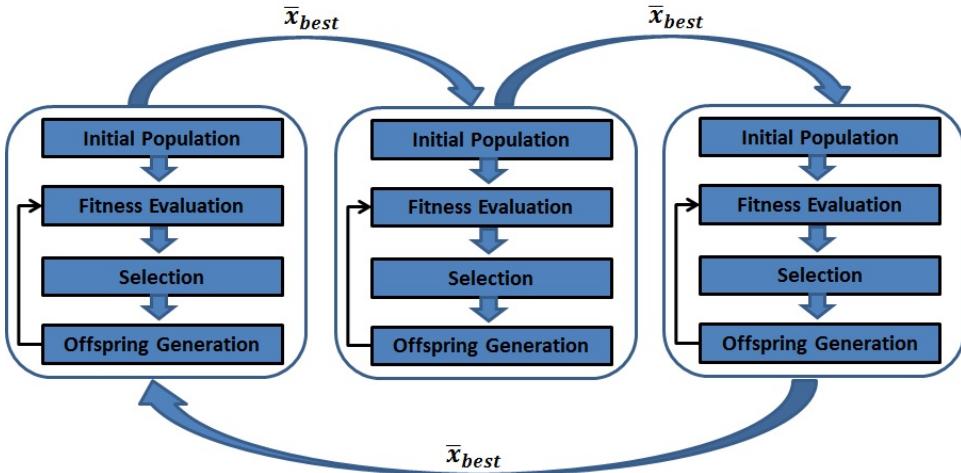


Figure 6.18. Simple round-robin parallel island architecture

6.11.4 Lunar Lander Results

These enhanced algorithms are run on Hamming using 256 processors evenly spread across eight compute nodes (32 processors per node). Initially, the USA-ES is compared with the RSA-ES on the lunar lander problem using 30 equally-spaced time nodes. Table 6.22 describes the settings used for this experiment. In this comparison experiment, each algorithm is run 10 times for 10^6 generations so that an average performance can be obtained. It can be seen in Figure 6.19 and in Table 6.23 that the USA-ES outperforms the RSA-ES and is quickly able to find feasible solutions to the optimal control problem. Despite using a large number of processors, covariance resets, and 10^6 generations, the RSA-ES is unable to find feasible solutions to the problem on any of the 10 trials. However, the USA-ES is able to find feasible, near-optimal solutions for all 10 trials.

Generations	Migrations	Processors	p	Recombination	Reset Tolerance	Trials
10^6	256	256	10^3	Logarithmic	10^{-6}	10

Table 6.22. USA-ES vs. RSA-ES parameters for the lunar lander problem

Method	Nodes	ρ	λ	t_f	$f(\bar{x}^*)$	Run Time (hours)	$f(\bar{x}^*)$ Error
USA-ES	30	31	63	1.446441	1.420873	0.12	$3.8 \times 10^{-2} \%$
RSA-ES	30	31	63	2.143810	5,213.153	0.23	$3.7 \times 10^5 \%$

Table 6.23. USA-ES vs. RSA-ES solutions to the lunar lander problem

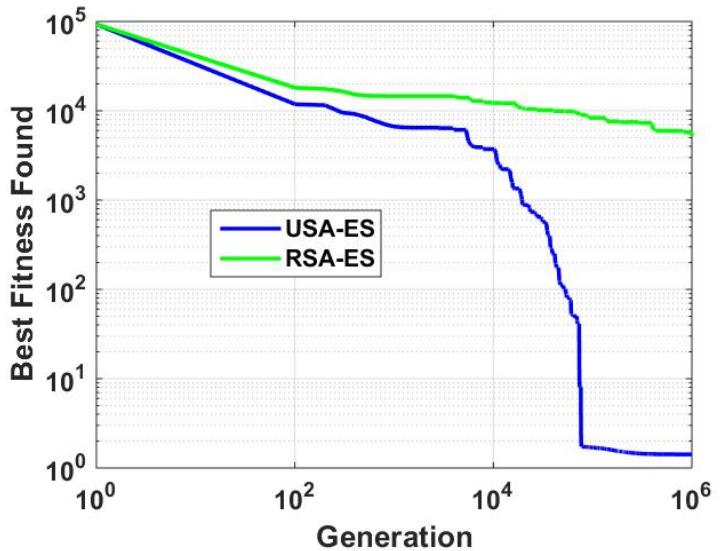


Figure 6.19. USA-ES vs. RSA-ES lunar lander problem average fitness progression

Next, the number of generations are increased along with the number of time nodes to see how accurately, the USA-ES is able to solve the lunar lander problem when compared to the known optimal solution to the problem. Table 6.24 describes the specific settings that are used to solve the problem and Table 6.25 summarizes the results. Figure 6.20 illustrates the results obtained from the USA-ES on the minimum propellant lunar lander problem. It can be seen that the USA-ES does a very good job of finding a solution that meets all of the constraints through the use of exact penalty functions on state variables and a simple repair technique on the control variable. The solution seems to improve and more closely approaches the optimal solution as the number of nodes or time-steps is increased, which is to be expected when using uniform time discretization. The last point on the control trajectory being set to zero on the USA-ES solution is expected given the method of numerical integration used. As one may recall, the integrator calculates the current state based on the previous control node. At the last node, the final state value is only dependent on the second to last control node. Given this, the algorithm obtains a better cost at no expense to the constraints by setting the last control node equal to zero as is seen in the results.

Generations	Migrations	Processors	p	Recombination	Reset Tolerance
5×10^6	256	256	10^3	Logarithmic	10^{-6}

Table 6.24. USA-ES parameters for the lunar lander problem

Method	Nodes	ρ	λ	t_f	$f(\bar{x}^*)$	Run Time (hours)	$f(\bar{x}^*)$ Error
USA-ES	30	31	63	1.444977	1.420317	0.5011	$1.2 \times 10^{-3} \%$
USA-ES	60	61	123	1.420438	1.420295	1.7409	$3.5 \times 10^{-4} \%$
USA-ES	80	81	163	1.414455	1.420299	3.2641	$7.0 \times 10^{-5} \%$

Table 6.25. USA-ES solutions to the lunar lander problem

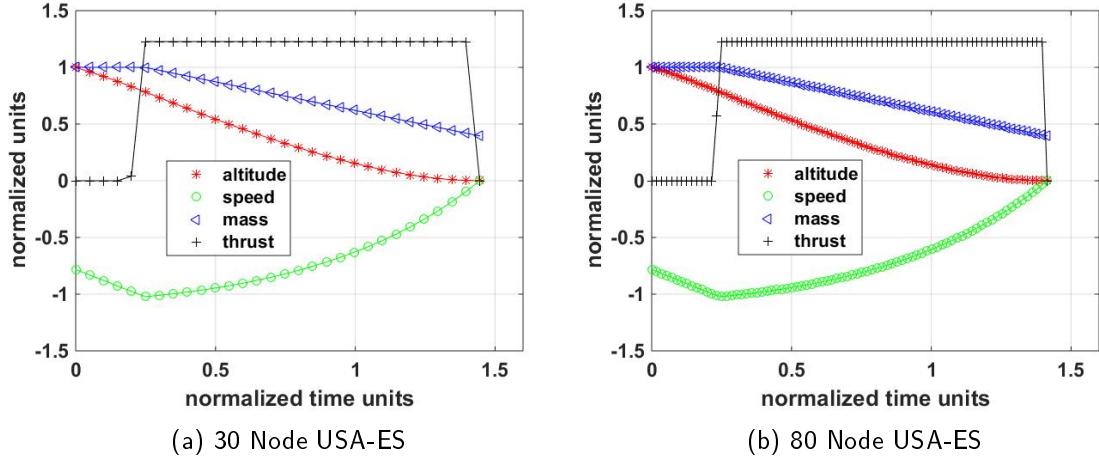


Figure 6.20. USA-ES solutions to the lunar lander problem

6.12 Conclusion

The results of this chapter illustrate how deterministic sampling has the potential to significantly improve performance of ESs. It has been shown through empirical examples that sigma points and sparse Gauss-Hermite points represent statistically meaningful sampling points that can be applied to more efficiently predict fitness landscapes in these types of stochastic algorithms. A simple covariance adaptation concept is utilized to overcome the loss of exploration capability that is achieved by having tails on a standard Gaussian distribution. These fairly simplistic ESs are able to significantly improve performance when

compared to the RSA-ES. They also compete nicely with the state-of-the-art CMA-ES and even outperforms this algorithm on some of the problems. Analytical gradients of the benchmark test problems are used to verify that the USA-ES is very effective at finding a local minimum with every run, and is able to find the globally optimal solution in many cases. The global search capability is further enhanced through the use of parallel processing by implementing a parallel island technique. Lastly, an optimal control problem is used to demonstrate the potential application and transition toward applying this USA-ES concept to real-world problems. This deterministic sampling concept produces very accurate results with the utilization of covariance resets and parallel processing. The introduction and development of the USA-ES and GHSA-ES is a unique and powerful contribution that has been shown here to increase optimization search performance on a number of challenging optimization problems and is a promising new development applicable to many different algorithms. The next chapter carries this idea of deterministic sampling even further by fully utilizing both the nodes and associated weights to intelligently adapt both the mean and covariance of search distributions within the construct of a state-of-the-art ES.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 7: Unscented Sampling In NESs

7.1 Introduction

The results from Chapter 6 are very encouraging, but are incomplete in a couple of ways. First of all, neither the USA-ES or the GHSA-ES fully utilize the information contained within the respective abscissas used for each algorithm as they only make use of the points and neglect the associated weights. Second, these algorithms only utilize the deterministic sampling to adapt the distribution mean and adapt the covariance through a predefined “cooling” schedule that is invariant to the sampling results and objective function values. This covariance adaptation scheme, while shown to be fairly successful on multimodal problems, is only one solution to the problem. A more intelligent scheme that is able to adapt the covariance as a function of the fitness landscape could likely further enhance algorithm performance. Given the promising empirical results shown in Chapter 6 regarding both algorithm efficiency and accuracy, this chapter provides several unique contributions to the ES community by further developing the concept of incorporating deterministic sampling techniques into state-of-the-art ESs. More specifically, the NES and xNES, described in Sections 5.13 and 5.14, are modified and developed to fully leverage sigma points. These new NES variants evolve both the mean and covariance matrix of the n-dimensional, normal search distributions by making use of the parametrized Gaussian nodes and their associated weights.

This chapter introduces the technique for using deterministic sampling within the construct of a NES. From there, it investigates the accuracy of these new methods by comparing analytically solved search gradients with estimates generated from both random and deterministic methods. Next, the application of deterministic sampling within the construct of an xNES is investigated and applied to increase algorithm accuracy, speed, and robustness. This new algorithm is then compared with the standard xNES along with the USA-ES on the set of highly multimodal test functions outlined in Table 6.1. From there, a new form of global optimization algorithm that utilizes the uxNES is developed and applied to a couple of the more challenging test problems. Lastly, the uxNES is applied to the lunar lander

problem to investigate its effectiveness and applicability to optimal control problems. A summary paper of the work in this chapter has been submitted for publication and will be presented at the AIAA Space 2016 Conference in Long Beach, CA [204].

7.2 unscented natural evolution strategy (uNES)

The use of a NES as the platform to further develop and investigate deterministic sampling within ESs quickly becomes appealing for several reasons. First of all, the NES appears to somewhat bridge the gap between stochastic and gradient-based optimization methods given that it uses stochastic methods to approximate search gradient information that is then used to update the ES parameters. Next, the NES variants are more eloquent and straightforward than many of the other state-of-the-art ESs, such as the CMA-ES. The logic and theory of how these algorithms work is fairly well-developed. This provides the theoretical background and building blocks that allow for a meaningful integration of deterministic sampling as part of the algorithm. Lastly, NESs have been shown to be fairly competitive in terms of solution accuracy and convergence speeds, though it must be noted that the CMA-ES has been shown to outperform conventional NESs in several instances [49], [191].

7.2.1 Search Gradients Using Unscented Sampling

This chapter introduces a new technique for determining the search gradient through the incorporation of unscented points. The standard NES search gradient technique described in Section 5.13, utilizes a set of points that have been randomly sampled from a normal distribution over a fitness landscape. This section investigates the use of deterministic sampling methods such as 3rd order sigma points and their associated weights to determine the expectations and resultant search gradients outlined in Section 5.13. It must be noted that the $2n + 1$ sigma points defined in Equation (6.2) in Section 6.3 also have a set of associated sigma weights, $W_{\sigma,i}$ that are calculated using Equation (7.1) [197].

$$W_{\sigma,i} = \begin{cases} \frac{\kappa}{(n+\kappa)} & \text{if } i = 1 \\ \frac{1}{2(n+\kappa)} & \text{else} \end{cases} \quad (7.1)$$

$i = [1, \dots, 2n + 1]$

With these sigma points and weights, a very similar approach as the one described in Section 5.13 can be used to more accurately determine the search gradient. Throughout this chapter, χ_i , represents the i th sigma point, $W_{\sigma,i}$ is the i th sigma weight value, n is the dimension of the search domain, μ represents the distribution mean, and κ is a user-defined parameter that controls the spread of the points. It must also be noted that this form of the sigma point calculation initially finds χ_i for a normal distribution with zero mean and an identity matrix for the covariance. Lastly, $\bar{\chi}_i$ is the set of sigma points that have been translated by a given mean μ and scaled by the square root of the covariance \sqrt{C} through an affine transformation $\bar{\chi}_i = \mu + \sqrt{C}\chi_i$.

Exact calculations of the mean and covariance for a Gaussian distribution can be recovered from the sigma points and associated weights using the methodology illustrated in Equation (7.2) [193]. This idea will be leveraged to more accurately solve for the search gradients within the construct of a NES, as they are each using quadrature to calculate Gaussian expectations. In Equation (7.2), both the mean, μ , and covariance, C , can be exactly calculated using 3rd order sigma points for a Gaussian distribution.

$$\begin{aligned} \mu &= \sum_{i=1}^{2n+1} W_{\sigma,i} \bar{\chi}_i \\ C &= \sum_{i=1}^{2n+1} W_{\sigma,i} (\bar{\chi}_i - \mu)(\bar{\chi}_i - \mu)^T \end{aligned} \quad (7.2)$$

As discussed in Section 5.13.1, the major component of an NES is finding the gradient of the expected fitness value, $J(\theta)$, with respect to the distribution mean and covariance, referred to as a search gradient and indicated by $\nabla_{\theta}J(\theta)$. It can be seen that the calculation of this search gradient is found by solving the expectation described in Equation (7.3). This is important given that the sigma points, and sparse, probabilistic Gauss-Hermite points are

well-suited for solving expectations of this form with Gaussian densities. In Equation (7.3), \bar{x} is an n-dimensional offspring solution vector, $f(\bar{x})$ is the objective function evaluated at \bar{x} , and $\nabla_\theta J(\theta)$ is the gradient of expected fitness with respect to the Gaussian search distribution parameters $\theta = [\mu, C]$. In the last line of Equation (7.3), it can be seen that the approximation of $\nabla_\theta J(\theta)$ is simply another form of Gaussian expectation that can be solved for using a quadrature technique with sigma points and associated weights.

$$\begin{aligned}\nabla_\theta J(\theta) &= \int [f(\bar{x})\nabla_\theta \log \pi(\bar{x}|\theta)] \pi(\bar{x}|\theta) d\bar{x} \\ &= \mathbb{E}_\theta [f(\bar{x})\nabla_\theta \log \pi(\bar{x}|\theta)]\end{aligned}\tag{7.3}$$

The standard NES utilizes Equation (7.4) to approximate the search gradient of expected fitness w.r.t. θ using random sample points.

$$\begin{aligned}\nabla_\theta J(\theta) &\approx \frac{1}{\lambda} \sum_{k=1}^{\lambda} f(\bar{x}^{(k)}) \nabla_\theta \log \pi(\bar{x}^{(k)}|\theta) \\ \text{where: } & \\ \nabla_\mu \log \pi(\bar{x}^{(k)}|\theta) &= C^{-1} (\bar{x}^{(k)} - \mu) \\ \nabla_C \log \pi(\bar{x}^{(k)}|\theta) &= \frac{1}{2} C^{-1} (\bar{x}^{(k)} - \mu) (\bar{x}^{(k)} - \mu)^T C^{-1} - \frac{1}{2} C^{-1}\end{aligned}\tag{7.4}$$

It is known that accurate solutions for expectations of the form described in Equation (7.3) can be obtained through the use of quadrature techniques coupled with the appropriate choice of nodes and associated weights. Given that the expectation in Equation (7.3) is using a Gaussian probability density function, the use of 3rd order sigma points and their associated weights are appropriate for approximating Equation (7.3) with a relatively small number of sample points. In theory, this simple quadrature technique should produce solutions that would require a significantly larger number of randomly selected sample points to match in terms of accuracy. Equation (7.5) illustrates how the search gradient can be modified to incorporate the 3rd order sigma points, or sparse Gauss-Hermite abscissas, to generate a more statistically meaningful, and hopefully more accurate expectation and

thus search gradient calculation.

$$\nabla_{\theta} J(\theta) \approx \sum_{k=1}^{\lambda} W_{\sigma,k} f(\bar{\chi}_k) \nabla_{\theta} \log \pi(\bar{\chi}_k | \theta)$$

where:

$$\nabla_{\mu} \log \pi(\bar{\chi}_k | \theta) = C^{-1}(\bar{\chi}_k - \mu)$$

$$\nabla_C \log \pi(\bar{\chi}_k | \theta) = \frac{1}{2} C^{-1}(\bar{\chi}_k - \mu)(\bar{\chi}_k - \mu)^T C^{-1} - \frac{1}{2} C^{-1}$$
(7.5)

Prior to further developing this algorithm to include a number of steps that increase algorithm robustness and global search capability, a set of experiments is warranted to develop an understanding of the advantages and performance enhancements that can be gained from utilizing the proposed method for calculating search gradients.

7.3 Accuracy of $\nabla_{\mu} J(\mu, \sigma)$ Search Gradient Estimates

The search gradient being calculated within an NES is the gradient of the expected fitness with respect to both the covariance and mean when considering normal distributions. This section investigates how closely the various techniques are able to estimate the gradient of expected fitness on sample functions with respect to the mean of a Gaussian distribution, μ . It must be noted that σ is the symbol used for the standard deviation.

7.3.1 $\nabla_{\mu} J(\mu, \sigma)$ sensitivity to function order

Two simple functions are explored in this experiment in order to determine how the order of an objective function impacts the accuracy of the various search gradient estimation methods. The first case investigates the objective function $f(x) = x^2$ where the analytical calculation of the gradient of the expected fitness, $J(\mu, \sigma)$, for the case of $\mu = 0$ $\sigma = 1$ is illustrated in Equation (7.7). Equation (7.6) is the analytical expression for the search gradient w.r.t. μ as a result of the log likelihood trick previously outlined in Equation (5.25).

$$\frac{\partial J(\mu, \sigma)}{\partial \mu} = \int_{-\infty}^{\infty} f(x) \frac{\partial \log(\pi(x|\mu, \sigma))}{\partial \mu} \pi(x|\mu, \sigma) dx$$
(7.6)

Next, the 1-D Gaussian density function along with the natural logarithm of the Gaussian density function are defined for clarity.

$$\pi(x|\mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)}$$

$$\log(\pi(x|\mu, \sigma)) = -\log(\sigma) - \frac{1}{2} \log(2\pi) - \frac{(x-\mu)^2}{2\sigma^2}$$

From here, the partial derivative of the Gaussian log density function w.r.t. μ is determined.

$$\frac{\partial \log(\pi(x|\mu, \sigma))}{\partial \mu} = \frac{x-\mu}{\sigma^2}$$

This equation is then further simplified by setting $\mu = 0$.

$$\left. \frac{\partial \log(\pi(x|\mu, \sigma))}{\partial \mu} \right|_{\mu=0} = \frac{x}{\sigma^2}$$

Next, the objective function $f(x)$, the partial derivative of the Gaussian log density function $\frac{\partial \log(\pi(x|\mu, \sigma))}{\partial \mu}$, and the Gaussian density function $\pi(x|\mu, \sigma)$ are all substituted back into Equation (7.6) resulting in Equation (7.7).

$$\frac{\partial J(\mu = 0, \sigma)}{\partial \mu} = \frac{1}{\sigma^3 \sqrt{2\pi}} \int_{-\infty}^{\infty} x^3 e^{\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)} \partial x \quad (7.7)$$

It must be noted that indefinite integrals of odd functions are equal to zero. In the case of $x^3 e^{\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)}$, it is an odd function multiplied by an even function, resulting in an odd function making the indefinite integral equal to zero. This results in an analytical search gradient that is equal to zero for this case, as depicted by Equation (7.8).

$$\frac{dJ(\mu = 0, \sigma = 1)}{d\mu} = 0.0 \quad (7.8)$$

This same exercise is performed on a 9th order function $f(x) = x^9$. This investigation is done for completeness to verify that the higher order Gauss-Hermite points are able to still accurately solve for higher order search gradients. The analytical equation for a 1D search gradient w.r.t. μ for a 9th order objective function evaluated at $\mu = 0$ is depicted in

(Equation 7.9).

$$\frac{\partial J(\mu = 0, \sigma)}{\partial \mu} = \frac{1}{\sigma^3 \sqrt{2\pi}} \int_{-\infty}^{\infty} x^{10} e^{\left(-\frac{(x)^2}{2\sigma^2}\right)} \partial x \quad (7.9)$$

Given that this is no longer an odd function, an integral table is used where the general form for this type of integral where the exponential, q , is an even integer is depicted in Equation (7.10). It is important to note that the $()!!$ operator is a double factorial where only the odd terms are multiplied e.g. $(5)!! = 1 \cdot 3 \cdot 5 = 15$.

$$\int_{-\infty}^{\infty} z^q e^{-\alpha z^2} \partial z = \frac{(2k - 1)!! \sqrt{\pi}}{2^k \alpha^k \sqrt{\alpha}} \text{ where: } k =^q /_2 \text{ and } \alpha = \frac{1}{2\sigma^2} \quad (7.10)$$

A resultant search gradient for the 9th order function with $\mu = 0$ is obtained after applying this general definition to Equation (7.9).

$$\frac{\partial J(\mu = 0, \sigma)}{\partial \mu} = \frac{1}{\sigma^3 \sqrt{2\pi}} \left[\frac{945 \sqrt{\pi}}{32 \left(\frac{1}{2\sigma^2} \right)^{5.5}} \right]$$

Lastly, this gradient is evaluated for the case where $\sigma = 1$ producing the analytical result depicted in Equation (7.11).

$$\frac{\partial J(\mu = 0, \sigma = 1)}{\partial \mu} = \frac{1}{\sqrt{2\pi}} \left[\frac{945 \sqrt{\pi}}{32 \left(\frac{1}{2} \right)^{5.5}} \right] = 945 \quad (7.11)$$

The random point error curve in Figure 7.1 represents the average magnitude of the error between the true search gradient and the estimated gradient. These average error magnitudes represent the average error norm obtained from estimating the search gradient with a given number of points 100 times. The number of sample points is then increased and this process is repeated to generate the search gradient error curve for the random sampling technique. The other methods are invariant from one run to the next, as they are deterministically chosen. It must be noted that some of the methods have errors that are exactly equal to zero. Given that these are plotted on logarithmic scales, they do not show up on the plot but are still depicted in the legend for completeness. As indicated in Equation (7.7), the calculation of the search gradient for the function $f(x) = x^2$ ends up being a 3rd order

equation. Given this, it is not surprising to see in Figure 7.1a and Table 7.1 that all of the deterministic methods that are 3rd order accurate or higher have an error norm of zero or within machine precision of zero. It is known that the integration error associated with Monte Carlo integration techniques like the ones used to estimate search gradients with random sampling is expected to improve on the order of approximately $\mathcal{O}\left(\frac{1}{\sqrt{\lambda}}\right)$, so the slope of the error curve on a log-log plot should be roughly $-1/2$ [200].

It can be seen in Figure 7.1b and Table 7.1, that both the 11th ($l=6$) and 15th ($l=8$) order Gauss-Hermite points are able to exactly calculate the search gradient for the 10th order expectation of the function $f(x) = x^9$, as would be expected. It is interesting to note that the randomly sampled points can become more accurate than the 3rd order ($l=2$) and 7th order ($l=4$) Gauss-Hermite points when a sufficient number of samples are taken for this 1-D example case where $\mu = 0$ $\sigma = 1$. It must be noted that when using the 3rd order sigma points with a κ value of zero, it is essentially the same as only using two points in this 1-D case. With this, it can be seen that the case with a non-zero κ value produces better results on the higher order function. These simple 1-D cases are used to make the analytical calculations a little less cumbersome, but are not entirely representative of a typical problem where the algorithm will likely be working in higher dimensions and with a non-zero mean. Given this, further analysis is needed to get a more realistic search gradient comparison for the types of problems that will be considered in this chapter.

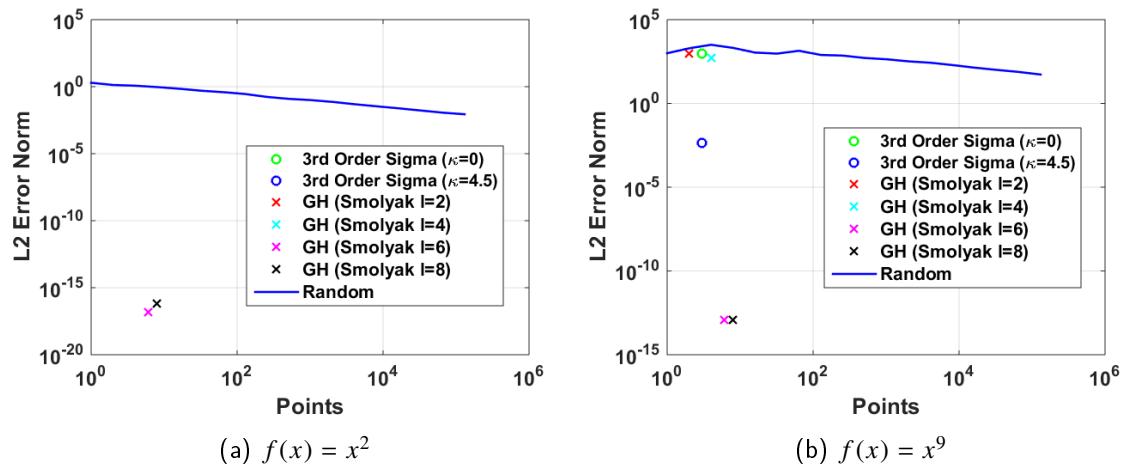


Figure 7.1. Accuracy of $\nabla_\mu J(\mu, \sigma)$ on the 1-D functions $f(x) = x^2$ and $f(x) = x^9$ ($\mu = 0$, $\sigma = 1$)

Method	$f(x) = x^2$ Error	$f(x) = x^9$ Error
Random (131,072 points)	0.01	53.9609
3rd Order Sigma $\kappa = 0$ (3 points)	0.0	944
3rd Order Sigma $\kappa = 4.5$ (3 points)	0.0	0.00452
3rd Order G-H Smolyak (2 points)	0.0	944
7th Order G-H Smolyak (4 points)	0.0	504
11th Order G-H Smolyak (6 points)	1.4×10^{-17}	1.1×10^{-13}
15th Order G-H Smolyak (8 points)	6.2×10^{-17}	1.1×10^{-13}

Table 7.1. Accuracy of $\nabla_\mu J(\mu, \sigma)$ on the 1-D functions $f(x) = x^2$ and $f(x) = x^9$ ($\mu = 0, \sigma = 1$)

7.3.2 Impact of varying σ on accuracy of $\nabla_\mu J(\mu, \sigma)$

Next, the same process is used to find the analytical search gradient of the function $f(x) = x^5$ with arbitrary values of μ and σ . As seen in Equation (7.12), a similar process to the previous case is followed, only this time a change of variables is required. With the result that follows, the analytical solution to the search gradient with respect to μ for the function $f(x) = x^5$ can now be found as a function of both μ and σ . This allows for a trade study to determine the effects that varying each parameter has on the search gradient error norms for the various methods.

Without assigning specific values to μ and σ , the derivative of the Gaussian log density function w.r.t. μ is depicted as follows.

$$\frac{\partial \log(\pi(x|\mu, \sigma))}{\partial \mu} = \frac{x - \mu}{\sigma^2}$$

Given this, Equation (7.12) depicts the function for the search gradient w.r.t. μ for generic values of μ and σ .

$$\frac{\partial J(\mu, \sigma)}{\partial \mu} = \frac{1}{\sigma^3 \sqrt{2\pi}} \int_{-\infty}^{\infty} x^5 (x - \mu) e^{\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)} dx \quad (7.12)$$

From this point, a change of variables is warranted where $z = (x - \mu)$. With this $\frac{\partial z}{\partial x} = 1 \rightarrow \partial z = \partial x$ and $x = z + \mu$. Using these newly defined relations, Equation (7.12) takes on a

new form as outlined in Equation (7.13).

$$\frac{\partial J(\mu, \sigma)}{\partial \mu} = \frac{1}{\sigma^3 \sqrt{2\pi}} \int_{-\infty}^{\infty} (z + \mu)^5 z e^{\left(-\frac{z^2}{2\sigma^2}\right)} dz \quad (7.13)$$

Next, the term $(z + \mu)^5 z$ is expanded to produce the following result.

$$(z + \mu)^5 z = \mu^5 z + 5\mu^4 z^2 + 10\mu^3 z^3 + 10\mu^2 z^4 + 5\mu z^5 + z^6$$

As previously mentioned, the odd terms result in indefinite integrals that go to zero. Given this, only the even terms need to be considered for the analytical calculation of this search gradient.

$$(z + \mu)^5 z_{\text{even terms}} = 5\mu^4 z^2 + 10\mu^2 z^4 + z^6$$

Next, this is substituted back into Equation (7.13) producing the following result.

$$\frac{\partial J(\mu, \sigma)}{\partial \mu} = \frac{1}{\sigma^3 \sqrt{2\pi}} \int_{-\infty}^{\infty} (5\mu^4 z^2 + 10\mu^2 z^4 + z^6) e^{\left(-\frac{z^2}{2\sigma^2}\right)} dz$$

From here, this integral can be split up into several integrals that all fit the general form of an indefinite integral depicted in Equation (7.10). This result is outlined as follows.

$$\begin{aligned} \frac{\partial J(\mu, \sigma)}{\partial \mu} &= \frac{5\mu^4}{\sigma^3 \sqrt{2\pi}} \int_{-\infty}^{\infty} z^2 e^{\left(-\frac{z^2}{2\sigma^2}\right)} dz \\ &\quad + \frac{10\mu^2}{\sigma^3 \sqrt{2\pi}} \int_{-\infty}^{\infty} z^4 e^{\left(-\frac{z^2}{2\sigma^2}\right)} dz + \frac{1}{\sigma^3 \sqrt{2\pi}} \int_{-\infty}^{\infty} z^6 e^{\left(-\frac{z^2}{2\sigma^2}\right)} dz \end{aligned}$$

After applying the general form of this type of integral to find the solution, the result becomes the following:

$$\frac{\partial J(\mu, \sigma)}{\partial \mu} = \frac{5\mu^4}{\sigma^3 \sqrt{2\pi}} \left[\frac{\sqrt{\pi}}{2 \left(\frac{1}{2\sigma^2} \right)^{1.5}} \right] + \frac{10\mu^2}{\sigma^3 \sqrt{2\pi}} \left[\frac{3\sqrt{\pi}}{4 \left(\frac{1}{2\sigma^2} \right)^{2.5}} \right] + \frac{1}{\sigma^3 \sqrt{2\pi}} \left[\frac{15\sqrt{\pi}}{8 \left(\frac{1}{2\sigma^2} \right)^{3.5}} \right]$$

Lastly, this result can be further simplified to produce the analytical result for the search

gradient w.r.t. μ of the function $f(x) = x^5$, as depicted in Equation (7.14).

$$\frac{\partial J(\mu, \sigma)}{\partial \mu} = 5\mu^4 + 30\mu^2\sigma^2 + 15\sigma^4 \quad (7.14)$$

After looking at Table 7.2 and Figure 7.2, it can be seen that the higher order Gauss-Hermite points are behaving as expected where they are able to produce exact results, given that they are using quadrature to find a Gaussian-based expectation of a 6th order function. It is interesting to see that adjusting the standard deviation has an opposite effect on the random points as it does on the 3rd order deterministically chosen points. It can be seen that a decreasing σ results in more accurate solutions for the 3rd order points, but results in less accurate search gradients for the random points. In addition, the higher order sparse Gauss-Hermite points seem to lose some accuracy as the standard deviation is decreased, although they are still very accurate for all cases. The relative error values in Table 7.3 illustrate the accuracy of each method relative to the magnitude of the analytical search gradient for each case. Similar trends can be seen by looking at these numbers as well.

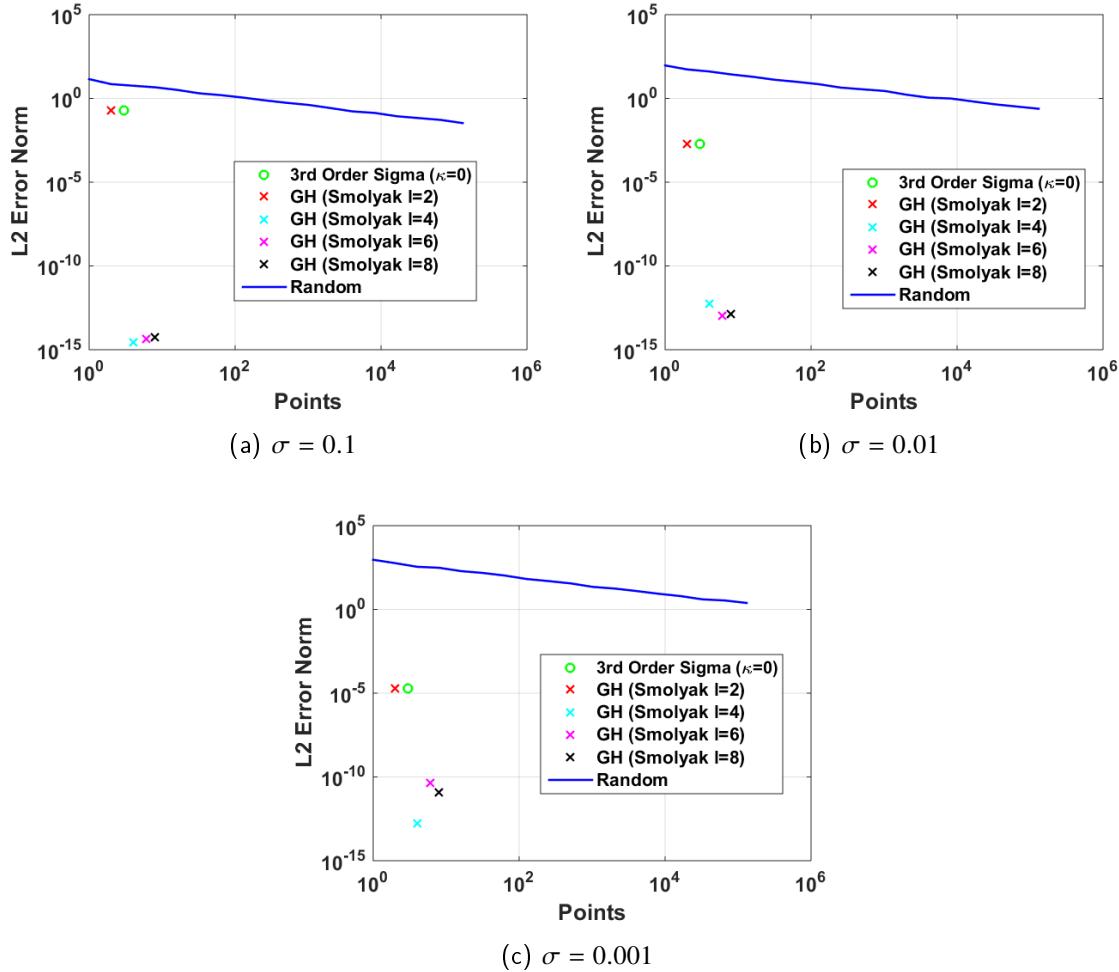


Figure 7.2. Accuracy of $\nabla_\mu J(\mu, \sigma)$ on the 1-D function $f(x) = x^5$ ($\mu = 1$ and varying σ)

Method	$(\sigma = 0.1)$	$(\sigma = 0.01)$	$(\sigma = 0.001)$
Random Best Avg. (131,072 points)	0.0336	0.2411	2.4256
3rd Order Sigma $\kappa = 0$ (3 points)	0.2014	0.0020	2.0×10^{-5}
3rd Order G-H Smolyak (2 points)	0.2014	0.0020	2.0×10^{-5}
7th Order G-H Smolyak (4 points)	2.7×10^{-15}	5.3×10^{-13}	1.7×10^{-13}
11th Order G-H Smolyak (6 points)	4.4×10^{-15}	1.1×10^{-13}	4.5×10^{-11}
15th Order G-H Smolyak (8 points)	5.3×10^{-15}	1.4×10^{-13}	1.2×10^{-11}

Table 7.2. $\nabla_\mu J(\mu, \sigma)$ error norms on the 1-D function $f(x) = x^5$ ($\mu = 1$ and varying σ)

Method	$(\sigma = 0.1)$	$(\sigma = 0.01)$	$(\sigma = 0.001)$
Random Best Avg. (131,072 points)	0.6%	4.8%	48.5%
3rd Order Sigma $\kappa = 0$ (3 points)	3.8%	0.04%	$4.0 \times 10^{-4}\%$
3rd Order G-H Smolyak (2 points)	3.8%	0.04%	$4.0 \times 10^{-4}\%$
7th Order G-H Smolyak (4 points)	$5.0 \times 10^{-14}\%$	$1.1 \times 10^{-11}\%$	$3.4 \times 10^{-12}\%$
11th Order G-H Smolyak (6 points)	$8.4 \times 10^{-14}\%$	$2.2 \times 10^{-12}\%$	$9.1 \times 10^{-10}\%$
15th Order G-H Smolyak (8 points)	$1.0 \times 10^{-13}\%$	$2.7 \times 10^{-12}\%$	$2.4 \times 10^{-10}\%$

Table 7.3. $\nabla_\mu J(\mu, \sigma)$ relative error on the 1-D function $f(x) = x^5$ ($\mu = 1$ and varying σ)

7.3.3 Impact of varying μ on accuracy of $\nabla_\mu J(\mu, \sigma)$

Next, a similar set of experiments is used to investigate the effect that varying the mean has on the accuracy of estimated search gradients with respect to μ . It must be noted that, within a typical NES, μ represents the location of the search distribution within the search domain and can take on many values throughout the progression of the algorithm. Given this, it is important to see how sensitive the various search gradient estimation techniques are to changes in μ . In these experiments, $\sigma = 0.1$ and the mean takes on various values as indicated in Figure 7.3 and Table 7.9. It can be seen that the error norms for all of the methods increase as the magnitude of the mean is increased. However, the error norms for the random points increase much more rapidly with respect to the magnitude of the mean than the other methods do. Table 7.5 illustrates the relative error with respect to the analytical solution for the various cases. The relative error illustrates the same reverse trend as was seen previously, where the random points get less accurate with regard to relative error as the magnitude of the mean is increased, and the 3rd order sigma points get increasingly accurate as the magnitude of the mean is increased. The higher order sigma points do not seem to follow a definite trend with respect to relative error as the magnitude of the mean is increased, as the differences are close to machine precision.

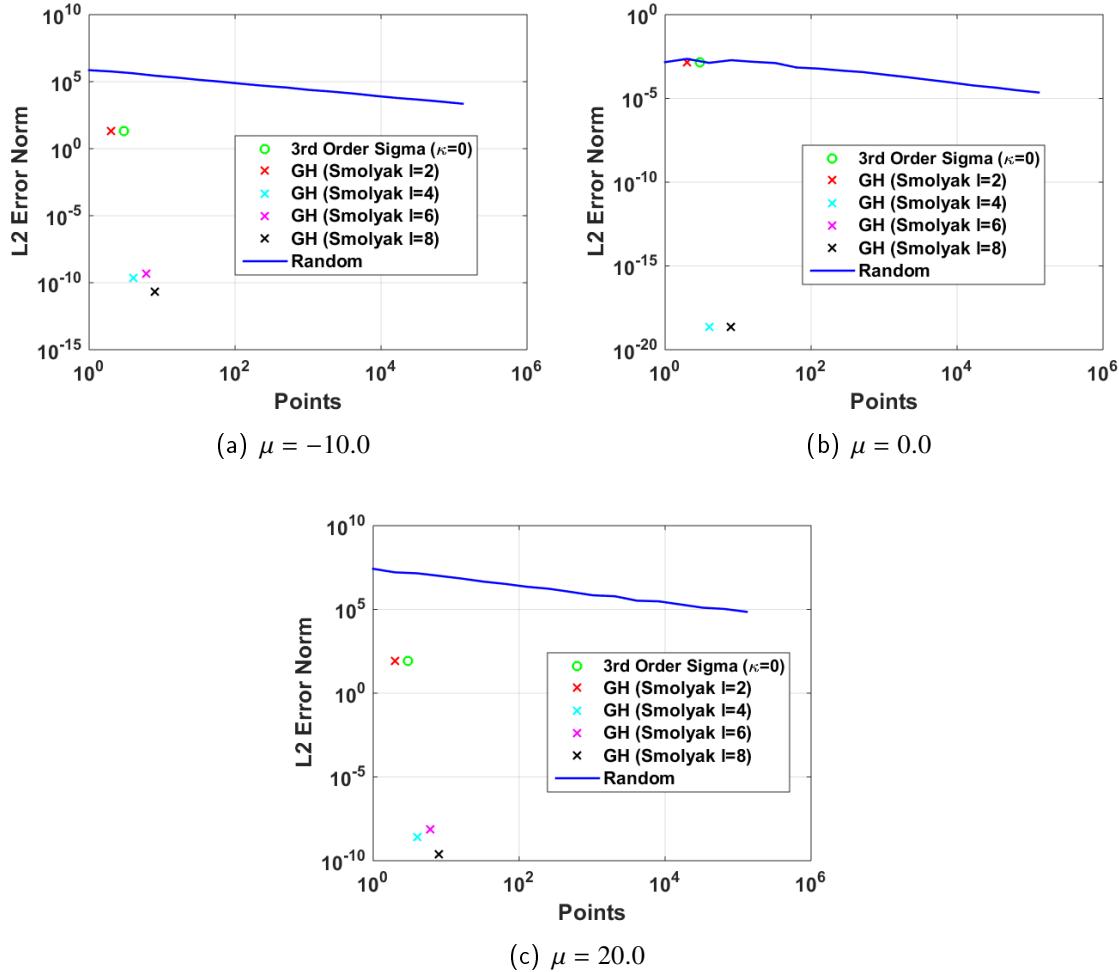


Figure 7.3. Accuracy of $\nabla_\mu J(\mu, \sigma)$ on the 1-D function $f(x) = x^5$ ($\sigma = 0.1$ and varying μ)

Method	$(\mu = -10.0)$	$(\mu = 0.0)$	$(\mu = 20)$
Random Best Avg. (131,072 points)	2256.3785	2.3×10^{-5}	71,658.4
3rd Order Sigma $\kappa = 0$ (3 points)	20.001	0.0014	80.0014
3rd Order G-H Smolyak (2 points)	20.001	0.0014	80.0014
7th Order G-H Smolyak (4 points)	2.2×10^{-10}	2.2×10^{-19}	2.6×10^{-9}
11th Order G-H Smolyak (6 points)	5.0×10^{-10}	0.0	7.5×10^{-9}
15th Order G-H Smolyak (8 points)	2.2×10^{-11}	2.2×10^{-19}	2.3×10^{-10}

Table 7.4. $\nabla_\mu J(\mu, \sigma)$ error norms on the 1-D function $f(x) = x^5$ ($\sigma = 0.1$ and varying μ)

Method	$(\mu = -10.0)$	$(\mu = 0.0)$	$(\mu = 20)$
Random Best Avg. (131,072 points)	4.5%	1.5%	9.0%
3rd Order Sigma $\kappa = 0$ (3 points)	0.04%	93.3%	0.01%
3rd Order G-H Smolyak (2 points)	0.04%	93.3%	0.01%
7th Order G-H Smolyak (4 points)	$4.4 \times 10^{-13}\%$	$1.4 \times 10^{-14}\%$	$3.2 \times 10^{-13}\%$
11th Order G-H Smolyak (6 points)	$1.0 \times 10^{-12}\%$	0.0%	$9.3 \times 10^{-13}\%$
15th Order G-H Smolyak (8 points)	$4.4 \times 10^{-14}\%$	$1.4 \times 10^{-14}\%$	$2.9 \times 10^{-14}\%$

Table 7.5. $\nabla_\mu J(\mu, \sigma)$ relative error on the 1-D function $f(x) = x^5$ ($\sigma = 0.1$ and varying μ)

7.3.4 Accuracy of $\nabla_\mu J(\mu, \sigma)$ for a 10-D problem

This experiment revisits the 5th order function using a multidimensional form $f(\bar{x}) = \sum_{i=1}^n x_i^5$, where $n = 10$. The function is evaluated at the point $\bar{x} = [1, 2, \dots, 10]$ with $C = 0.1 \times I$. Given that there are no cross terms in the covariance matrix, and the random variable is independent and identically distributed (i.i.d.), the Equation (7.14) can be used to calculate each component of the search gradient in each dimension independently as follows: $\nabla_\mu J(\mu, \sigma)_i = 5\mu_i^4 + 30\mu_i^2\sigma_{i,i}^2 + 15\sigma_{i,i}^4$ $i = [1, \dots, n]$. Figure 7.4 and Table 7.6 illustrate these results. It can be seen that the 3rd order points perform very favorably when compared to fairly large numbers of random points in the case where there is a non-zero mean and the dimension is larger than one. More specifically, the estimated search direction error, or angle between the true search gradient and estimated search gradient is off by only eight hundredths of a degree. This accuracy cannot be matched with random sampling, even after using 10^6 sample points. Due to the fact that the NESs utilize learning rate parameters, the direction error between the analytical search gradient and estimated search gradient is likely the most important metric for the purposes of a typical NES application. This is due to the fact that the learning rates alter the magnitude of the estimated search gradients to change the search progression speeds, however, the angles or search directions are not affected by these rates.

It must also be noted that the computational error due to roundoff is increasingly noticeable in the higher dimensional cases where there are significantly larger numbers of sample points. This effect was not as noticeable in the 1-D cases where the difference between each set of higher order Gauss-Hermite points was relatively small. Given this, it is best to

use the lowest order abscissas possible that still have a high enough accuracy to accurately integrate the given expectation order. The significant increase in search gradient accuracy when using the deterministic methods is the primary justification for employing the use of 3rd order sigma points, or higher order sparse Gauss-Hermite points, within the construct of the NES algorithms. This use of deterministic sampling to improve search gradient accuracy proves to be very useful when solving the various test problems throughout this chapter.

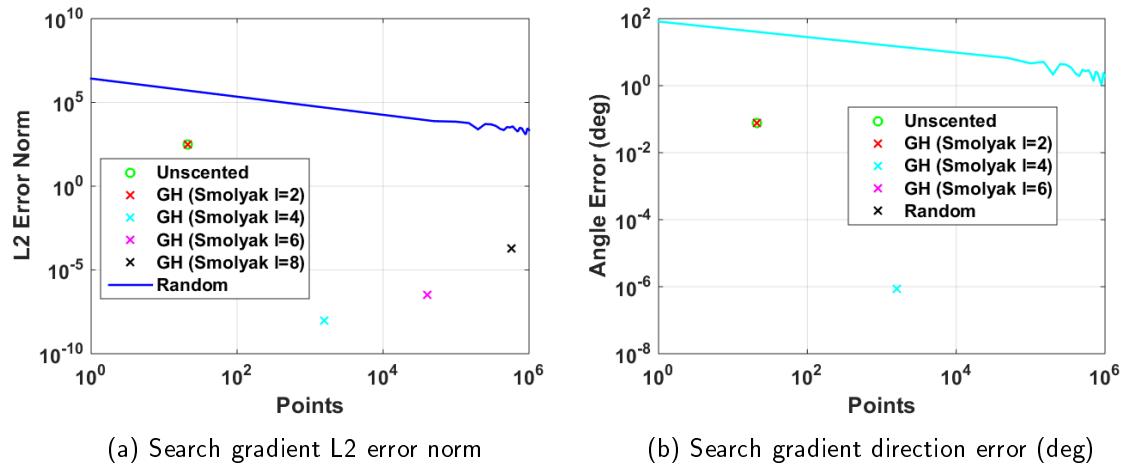


Figure 7.4. $\nabla_\mu J(\mu, \sigma)$ accuracy for the 10-D function $f(\bar{x}) = \sum_{i=1}^n x_i^5$ ($\bar{\mu} = [1, \dots, n]^T$ and $C = 0.1 \times I$)

Method	L2 Error Norm	Relative Error	Direction Error (deg)
Random (1,000,000 pts)	1205.83	1.85%	1.04
3rd Order Unscented $\kappa = -9$ (21 pts)	318.67	0.49%	0.08
3rd Order GH Smolyak (21 pts)	318.67	0.49%	0.08
7th Order GH Smolyak (1581 pts)	9.9×10^{-9}	$1.5 \times 10^{-11}\%$	8.5×10^{-7}
11th Order GH Smolyak (40,405 pts)	3.3×10^{-7}	$5.0 \times 10^{-10}\%$	0.0
15th Order GH Smolyak (581,385 pts)	1.9×10^{-4}	$2.9 \times 10^{-7}\%$	0.0

Table 7.6. $\nabla_\mu J(\mu, \sigma)$ accuracy on 10-D function $f(\bar{x}) = \sum_{i=1}^n x_i^5$ ($\bar{\mu} = [1, \dots, n]^T$ and $C = 0.1 \times I$)

7.4 Accuracy of $\nabla_\sigma J(\mu, \sigma)$ Search Gradient Estimates

Next, a similar sensitivity analysis considering the search gradient with respect to the standard deviation, $\nabla_\sigma J(x|\mu, \sigma)$ is conducted. A similar exercise must be performed in order to find the analytical solution to a search gradient w.r.t. σ on the function $f(x) = x^5$ for arbitrary values of μ and σ .

Equation (7.15) is the analytical expression for the search gradient w.r.t. σ as a result of the log likelihood trick previously outlined in Equation (5.25).

$$\frac{\partial J(\mu, \sigma)}{\partial \sigma} = \int_{-\infty}^{\infty} f(x) \frac{\partial \log(\pi(x|\mu, \sigma))}{\partial \sigma} \pi(x|\mu, \sigma) dx \quad (7.15)$$

The process is essentially the same as before, only this time the derivative of the log density function with respect to σ is used.

$$\frac{\partial \log(\pi(x|\mu, \sigma))}{\partial \sigma} = \frac{(x - \mu)^2}{\sigma^3} - \frac{1}{\sigma}$$

From this point, the search gradient w.r.t. σ for the function $f(x) = x^5$ is described by Equation (7.16) and is obtained by substituting the appropriate terms back into Equation (7.15).

$$\frac{\partial J(\mu, \sigma)}{\partial \sigma} = \frac{1}{\sigma^2 \sqrt{2\pi}} \int_{-\infty}^{\infty} x^5 \left(\frac{(x - \mu)^2}{\sigma^2} - 1 \right) e^{\left(-\frac{(x - \mu)^2}{2\sigma^2} \right)} dx \quad (7.16)$$

Next, a change of variables is used such that $z = (x - \mu)$. With this $\frac{\partial z}{\partial x} = 1 \rightarrow \partial z = \partial x$ and $x = z + \mu$. Using these newly defined relations, Equation (7.16) takes on a new form as outlined in Equation (7.17).

$$\frac{\partial J(\mu, \sigma)}{\partial \sigma} = \frac{1}{\sigma^2 \sqrt{2\pi}} \int_{-\infty}^{\infty} (z + \mu)^5 \left(\frac{z^2}{\sigma^2} - 1 \right) e^{\left(-\frac{z^2}{2\sigma^2} \right)} dz \quad (7.17)$$

From here, the term $(z + \mu)^5 \left(\frac{z^2}{\sigma^2} - 1 \right)$ is expanded to produce the following result.

$$\begin{aligned} (z + \mu)^5 \left(\frac{z^2}{\sigma^2} - 1 \right) &= \frac{z^7}{\sigma^2} - 10\mu^3 z^2 - 10\mu^2 z^3 - 5\mu z^4 - 5\mu^4 z - \mu^5 - z^5 \\ &\quad + \frac{5\mu z^6}{\sigma^2} + \frac{10\mu^2 z^5}{\sigma^2} + \frac{10\mu^3 z^4}{\sigma^2} + \frac{5\mu^4 z^3}{\sigma^2} + \frac{\mu^5 z^2}{\sigma^2} \end{aligned}$$

As previously mentioned, the odd terms result in indefinite integrals that go to zero. Given this, only the even terms need to be considered for the analytical calculation of this search gradient.

$$\left[(z + \mu)^5 \left(\frac{z^2}{\sigma^2} - 1 \right) \right]_{even \ terms} = -\mu^5 + \left(\frac{\mu^5}{\sigma^2} - 10\mu^3 \right) z^2 + \left(\frac{10\mu^3}{\sigma^2} - 5\mu \right) z^4 + \frac{5\mu z^6}{\sigma^2}$$

Next, this is substituted back into Equation (7.13) producing the following result. It can be seen that these integrals all fit the general form of an indefinite integral depicted in Equation (7.10).

$$\begin{aligned} \frac{\partial J(\mu, \sigma)}{\partial \sigma} = & -\frac{\mu^5}{\sigma^2 \sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-\frac{z^2}{2\sigma^2}} \partial z + \frac{\frac{\mu^5}{\sigma^2} - 10\mu^3}{\sigma^2 \sqrt{2\pi}} \int_{-\infty}^{\infty} z^2 e^{-\frac{z^2}{2\sigma^2}} \partial z \\ & + \frac{\frac{10\mu^3}{\sigma^2} - 5\mu}{\sigma^2 \sqrt{2\pi}} \int_{-\infty}^{\infty} z^4 e^{-\frac{z^2}{2\sigma^2}} \partial z + \frac{5\mu}{\sigma^4 \sqrt{2\pi}} \int_{-\infty}^{\infty} z^6 e^{-\frac{z^2}{2\sigma^2}} \partial z \end{aligned}$$

After applying the general form outlined Equation (7.13) to find the solution, the result becomes the following:

$$\begin{aligned} \frac{\partial J(\mu, \sigma)}{d\sigma} = & -\frac{\mu^5}{\sigma^2 \sqrt{2\pi}} \left[\frac{\sqrt{\pi}}{\left(\frac{1}{2\sigma^2} \right)^{0.5}} \right] + \frac{\frac{\mu^5}{\sigma^2} - 10\mu^3}{\sigma^2 \sqrt{2\pi}} \left[\frac{\sqrt{\pi}}{2 \left(\frac{1}{2\sigma^2} \right)^{1.5}} \right] \\ & + \frac{\frac{10\mu^3}{\sigma^2} - 5\mu}{\sigma^2 \sqrt{2\pi}} \left[\frac{3\sqrt{\pi}}{4 \left(\frac{1}{2\sigma^2} \right)^{2.5}} \right] + \frac{5\mu}{\sigma^4 \sqrt{2\pi}} \left[\frac{15\sqrt{\pi}}{8 \left(\frac{1}{2\sigma^2} \right)^{3.5}} \right] \end{aligned}$$

Lastly, this result can be further simplified to produce the analytical result for the search gradient w.r.t. σ of the function $f(x) = x^5$, as depicted in Equation (7.18).

$$\frac{\partial J(\mu, \sigma)}{\partial \sigma} = 20\sigma\mu (\mu^2 + 3\sigma^2) \quad (7.18)$$

7.4.1 Impact of varying σ on accuracy of $\nabla_\sigma J(\mu, \sigma)$

With the analytical expression for the search gradient w.r.t. σ , trades can now be performed to see how accurate each estimation method is. Figure 7.5, Table 7.7, and Table 7.8 illustrate the results obtained from the different search gradient of the function $f(x) = x^5$ estimation techniques for three different values of σ . Again, it can be seen that sparse Gauss-Hermite abscissas that have an order that is equal to or higher than 6th order are able to obtain machine precision accuracy. This is expected and provides more confidence that the analytical solution to this search gradient, along with the methodology used to estimate it are correct. With this, it can be seen that varying σ has little to no effect on the unscented techniques. This is slightly different than what was seen previously, where the error improved as σ was decreased. It can also be seen that the error norms and relative errors associated with the random sampling technique get worse as the value of σ is decreased. This trend is similar to what was seen in the previous experiments regarding the search gradient w.r.t. μ .

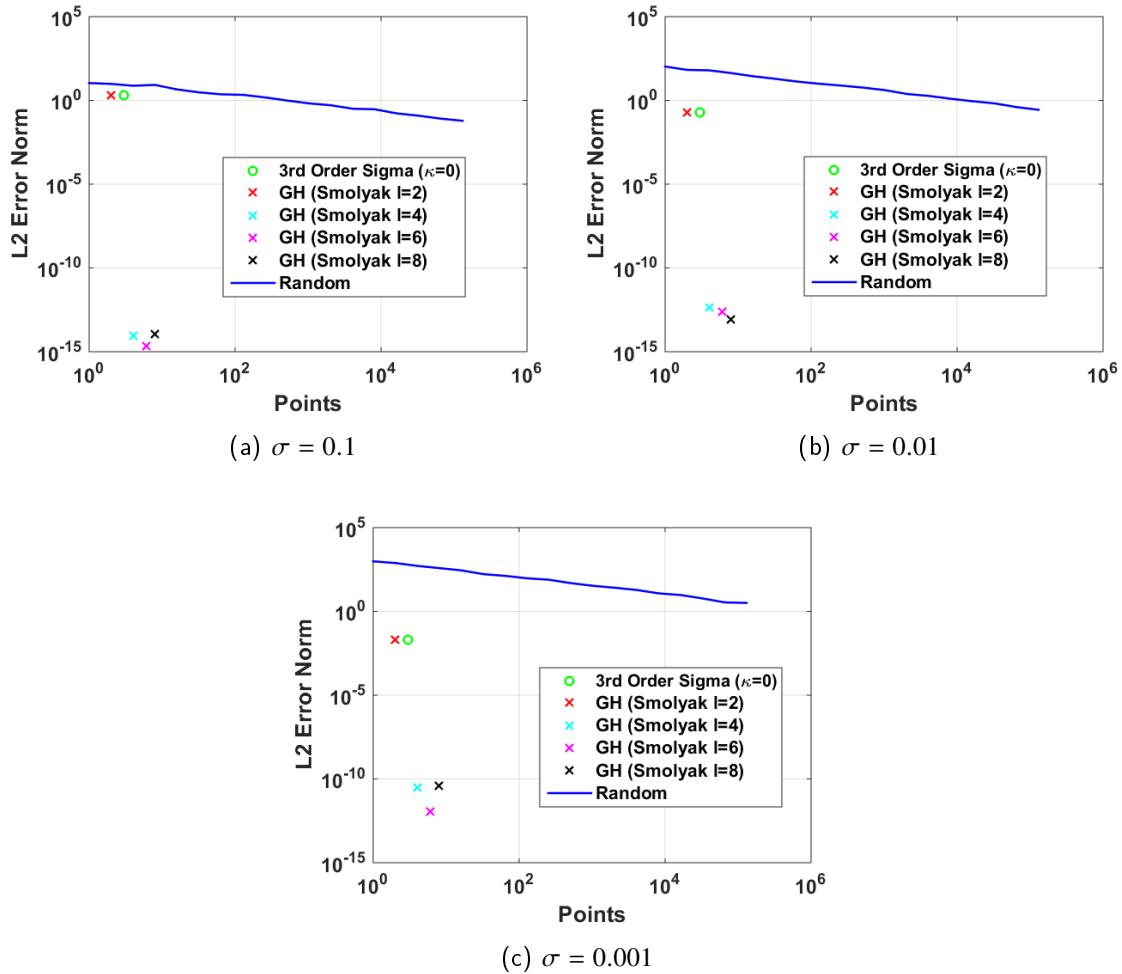


Figure 7.5. Accuracy of $\nabla_\sigma J(\mu, \sigma)$ estimates on the 1-D function $f(x) = x^5$ ($\mu = 1$ and varying σ)

Method	$(\sigma = 0.1)$	$(\sigma = 0.01)$	$(\sigma = 0.001)$
Random Best Avg. (131,072 points)	0.061	0.274	3.222
3rd Order Sigma $\kappa = 0$ (3 points)	2.06	0.200	0.02
3rd Order G-H Smolyak (2 points)	2.06	0.200	0.02
7th Order G-H Smolyak (4 points)	9.3×10^{-15}	4.3×10^{-13}	2.9×10^{-11}
11th Order G-H Smolyak (6 points)	2.2×10^{-15}	2.4×10^{-13}	1.1×10^{-12}
15th Order G-H Smolyak (8 points)	1.1×10^{-14}	8.6×10^{-14}	3.8×10^{-11}

Table 7.7. $\nabla_\sigma J(\mu, \sigma)$ error norms on the 1-D function $f(x) = x^5$ ($\mu = 1$ and varying σ)

Method	$(\sigma = 0.1)$	$(\sigma = 0.01)$	$(\sigma = 0.001)$
Random Best Avg. (131,072 points)	3%	137.1%	16,111.5%
3rd Order Sigma $\kappa = 0$ (3 points)	100%	100%	100%
3rd Order G-H Smolyak (2 points)	100%	100%	100%
7th Order G-H Smolyak (4 points)	$4.5 \times 10^{-13}\%$	$2.2 \times 10^{-10}\%$	$1.5 \times 10^{-7}\%$
11th Order G-H Smolyak (6 points)	$1.1 \times 10^{-13}\%$	$1.2 \times 10^{-10}\%$	$5.3 \times 10^{-9}\%$
15th Order G-H Smolyak (8 points)	$5.2 \times 10^{-13}\%$	$4.3 \times 10^{-11}\%$	$1.9 \times 10^{-7}\%$

Table 7.8. $\nabla_\sigma J(\mu, \sigma)$ relative error on the 1-D function $f(x) = x^5$ ($\mu = 1$ and varying σ)

7.4.2 Sensitivity of varying μ on accuracy of $\nabla_\sigma J(\mu, \sigma)$

A similar set of experiments are used to investigate the effect that varying the mean has on the accuracy of estimated search gradients with respect to σ . In these experiments, $\sigma = 0.1$ and the mean takes on various values as indicated in Figure 7.6 and Table 7.3. It can be seen that the error norms for all of the methods increase as the magnitude of the mean is increased. However, it can be seen that the error norms for the random points increases much more rapidly with respect to the magnitude of the mean than the other methods do. Table 7.10 illustrates the relative error with respect to the analytical solution for the various cases. The trends in relative error are a little more telling in how the various methods are behaving with respect to increasing the magnitude of μ . It is interesting to note that the relative error using random sampling significantly increases when the magnitude of μ is increased, whereas the relative error for the deterministic methods is constant, or close to constant.

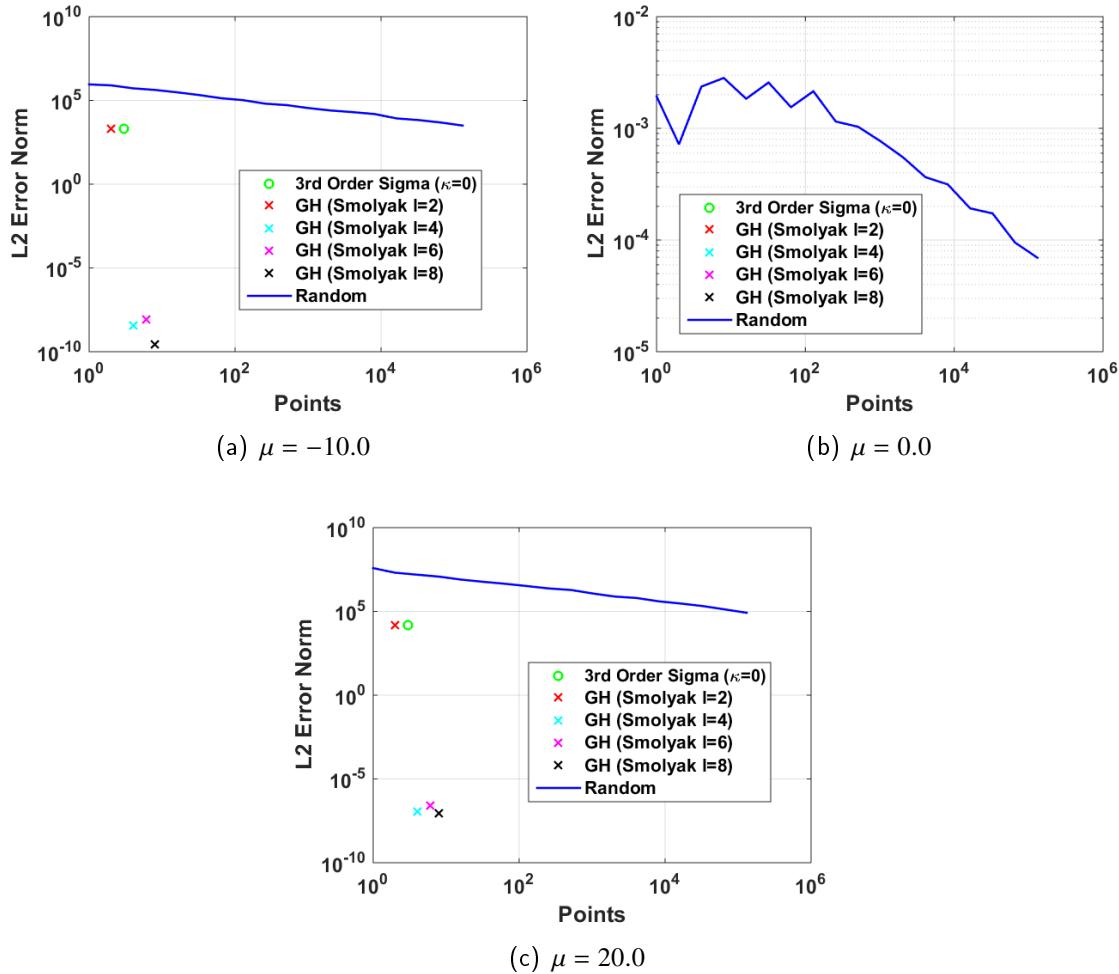


Figure 7.6. $\nabla_\sigma J(\mu, \sigma)$ accuracy on the 1-D function $f(x) = x^5$ ($\sigma = 0.1$ and varying μ)

Method	$(\mu = -10.0)$	$(\mu = 0.0)$	$(\mu = 20)$
Random Best Avg. (131,072 points)	3,152.6	6.9×10^{-5}	82,903.1
3rd Order Sigma $\kappa = 0$ (3 points)	2,000.6	0.0	16,001.2
3rd Order G-H Smolyak (2 points)	2,000.6	0.0	16,001.2
7th Order G-H Smolyak (4 points)	3.5×10^{-9}	0.0	1.1×10^{-7}
11th Order G-H Smolyak (6 points)	8.2×10^{-9}	0.0	2.6×10^{-7}
15th Order G-H Smolyak (8 points)	2.7×10^{-10}	0.0	9.4×10^{-8}

Table 7.9. $\nabla_\sigma J(\mu, \sigma)$ error norms on the 1-D function $f(x) = x^5$ ($\sigma = 0.1$ and varying μ)

Method	$(\mu = -10.0)$	$(\mu = 0.0)$	$(\mu = 20)$
Random Best Avg. (131,072 points)	157.6%	<i>Nan</i> %	518.1%
3rd Order Sigma $\kappa = 0$ (3 points)	100%	0.0%	100%
3rd Order G-H Smolyak (2 points)	100%	0.0%	100%
7th Order G-H Smolyak (4 points)	$1.7 \times 10^{-10}\%$	0.0%	$6.9 \times 10^{-10}\%$
11th Order G-H Smolyak (6 points)	$4.1 \times 10^{-10}\%$	0.0%	$1.7 \times 10^{-9}\%$
15th Order G-H Smolyak (8 points)	$1.3 \times 10^{-11}\%$	0.0%	$5.8 \times 10^{-10}\%$

Table 7.10. $\nabla_\sigma J(\mu, \sigma)$ relative error on the 1-D function $f(x) = x^5$ ($\sigma = 0.1$ and varying μ)

7.4.3 uNES Performance

As is illustrated in Section 7.2.1, more accurate search gradients can be obtained through the use of quadrature and deterministically chosen sample points and weights. Algorithm 3 illustrates the result of putting all of these pieces together. It can be seen that the search gradients, $[\nabla_\mu J(\theta), \nabla_C J(\theta)]$, and Fischer information matrices, $[F_{\mu^{(g)}}, F_{C^{(g)}}]$, are all determined using deterministic points, $\bar{\chi}$ and weights, W_σ . Lastly, the mean, μ , and covariance, C , are updated using the same technique as the standard NES. Algorithm 3 describes the uNES that uses the improved, unscented natural search gradient techniques within the construct of a standard NES. This algorithm does not use the fitness shaping techniques that are often used in many of the NES variants [41]. In many of the NES algorithms, the fitness values of the offspring are scaled or shaped with respect to their rank order. As an example, the standard NES often utilizes a logarithmic fitness shaping scheme,

similar to the CMA-ES, as described in Section 5.13.3.

Algorithm 3: uNES Algorithm

Initialize: $\mu^{(0)}$ = random uniform $\in [LB, UB]^{\mathbb{R}^n}$;

User-Defined Parameters: g_{max} , η_μ , η_C , and $C^{(0)}$;

$g = 0$;

while stopping criteria not satisfied **do**

Transform unscented points for given μ and C ;

$$\bar{\chi}_k = \mu^{(g)} + \sqrt{C^{(g)}} \chi_k;$$

Evaluate $f(\bar{\chi}_k)$ $k = [1, \dots, \lambda]$;

Determine search gradients;

$$\nabla_\mu J(\theta) \approx \sum_{k=1}^{\lambda} W_{\sigma,k} f(\bar{\chi}_k) \left[\frac{1}{C^{(g)}} (\bar{\chi}_k - \mu^{(g)}) \right];$$

$$\nabla_C J(\theta) \approx \sum_{k=1}^{\lambda} W_{\sigma,k} f(\bar{\chi}_k) \left[\frac{1}{2} \frac{1}{C^{(g)}} (\bar{\chi}_k - \mu^{(g)}) (\bar{\chi}_k - \mu^{(g)})^T \frac{1}{C^{(g)}} - \frac{1}{2} \frac{1}{C^{(g)}} \right];$$

Find Fischer matrices using unscented points;

$$F_{\mu^{(g)}} = \sum_{k=1}^{\lambda} W_{\sigma,k} \left(\frac{1}{C^{(g)}} (\bar{\chi}_k - \mu^{(g)}) \right) \left(\frac{1}{C^{(g)}} (\bar{\chi}_k - \mu^{(g)}) \right)^T;$$

$$F_{C^{(g)}} = \sum_{k=1}^{\lambda} W_{\sigma,k} \left(\frac{1}{2} \frac{1}{C^{(g)}} (\bar{\chi}_k - \mu^{(g)}) (\bar{\chi}_k - \mu^{(g)})^T \frac{1}{C^{(g)}} - \frac{1}{2} \frac{1}{C^{(g)}} \right) \\ \times \left(\frac{1}{2} \frac{1}{C^{(g)}} (\bar{\chi}_k - \mu^{(g)}) (\bar{\chi}_k - \mu^{(g)})^T \frac{1}{C^{(g)}} - \frac{1}{2} \frac{1}{C^{(g)}} \right)^T;$$

Update μ and C ;

$$\mu^{(g+1)} = \mu^{(g)} + \eta_\mu (F_{\mu^{(g)}})^{-1} \nabla_\mu J(\theta);$$

$$C^{(g+1)} = C^{(g)} + \eta_C (F_{C^{(g)}})^{-1} \nabla_C J(\theta);$$

$g = g + 1$;

end

The uNES outlined in Algorithm 3 is applied to a simple, 10-D parabolic function where $f(\bar{x}) = \sum_{i=1}^n x_i^2$. In addition, the standard version of an NES with random sampling and fitness shaping is also applied to the same problem using the same learning rates and initial covariance values. Figure 7.7 and Table 7.11 illustrates this simple comparison, where the increased search gradient accuracy appears to increase both convergence speed and accuracy for this simple case.

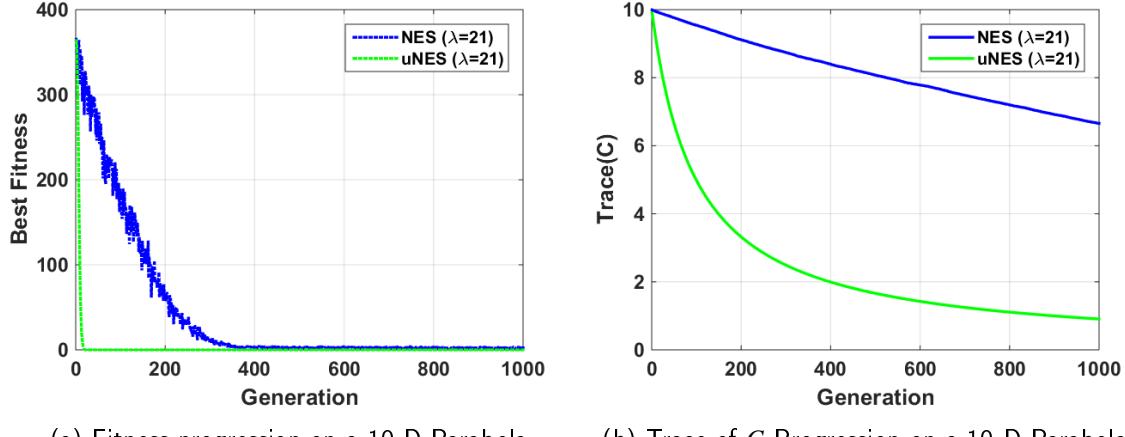


Figure 7.7. uNES/NES comparison on parabolic function

Figure	C_0	η_C	η_μ
7.7	I	$0.5 * \min\{1.0/n, 0.25\}$	1.0

Table 7.11. uNES/NES comparison parameters

This section has briefly shown the potential benefit of using deterministic sampling in the construct of an NES algorithm by observing search gradient estimation accuracy and optimization efficiency of a uNES on simple parabolic function. The next section investigates the application of deterministic sampling within the construct of an xNES. This exponential version of the NES proves to be more robust and efficient for both random sampling and deterministic sampling, and will therefore be developed and applied to the set of multi-modal benchmark functions introduced in Table 6.1. The standard NES is vulnerable to matrix inversion issues coupled with the potential to evolve the covariance matrix to values that are no longer positive definite. While the latter issue can potentially be avoided using several different techniques, the uxNES offers an elegant and computationally efficient approach [191].

7.5 uxNES

As introduced in Section 5.14, the exponential form of the xNES can be developed using a similar idea as what was done to develop the uNES. The exponential form of the NES

employs the use of a change of coordinates so that the covariance and mean are being updated in a natural, exponential space. This mapping forces the updated covariance matrix to always be positive definite, and it removes the requirement to calculate or invert a Fischer information matrix resulting in a faster and more robust algorithm [191]. However, in order to use the same discretization methods as were used in the uNES, it has to be shown that the underlying method for determining the search gradient is the same as depicted in Equation (7.3) where the same Gaussian expectation is being solved after it has been mapped to this new coordinate frame.

7.5.1 Relation to $\nabla_{\theta} J(\theta)$ Expectation

The following exercise develops the xNES equations from this original Gaussian-based expectation outlined in Equation (7.19) so it can be shown that the choice of 3rd order sigma points and probabilistic Gauss-Hermite points are both appropriate choices for deterministic sampling points used in obtaining more accurate search gradients within the construct of an xNES. Glasmachers et al. utilize an exponential mapping to avoid problems associated with updating the covariance matrix in terms of maintaining positive definiteness that has the following characteristics:

Let $\mathcal{S}_d := \left\{ M \in \mathbb{R}^{d \times d} \mid M^T = M \right\}$ and $\mathcal{P}_d := \left\{ M \in \mathcal{S}_d \mid v^T M v > 0 \text{ for all } v \in \mathbb{R}^d \setminus \{0\} \right\}$ denote the vector space of symmetric and the (cone) manifold of symmetric positive definite matrices, respectively. Then the exponential map $\exp : \mathcal{S}_d \rightarrow \mathcal{P}_d$, $M \mapsto \sum_{n=0}^{\infty} \frac{M^n}{n!}$ is a diffeomorphism: The map is bijective, and both \exp as well as its inverse map $\log : \mathcal{P}_d \rightarrow \mathcal{S}_d$ are smooth. [191]

For completeness, the log likelihood trick and resulting expectation to be solved is restated in Equation (7.19) where the offspring solution vectors, \bar{x} , are randomly selected from a Gaussian distribution. The remaining steps taken are conducted to show that the uxNES equations directly stem from this expectation, and can therefore be efficiently calculated

using the unscented sigma points and sparse Gauss-Hermite points.

$$\begin{aligned}
\nabla_{\theta} J(\theta) &= \nabla_{\theta} \int f(\bar{x}) \pi(\bar{x}|\theta) d\bar{x} \\
&= \int f(\bar{x}) \nabla_{\theta} \pi(\bar{x}|\theta) d\bar{x} \\
&= \int f(\bar{x}) \nabla_{\theta} \pi(\bar{x}|\theta) \frac{\pi(\bar{x}|\theta)}{\pi(\bar{x}|\theta)} d\bar{x} \\
&= \int [f(\bar{x}) \nabla_{\theta} \log \pi(\bar{x}|\theta)] \pi(\bar{x}|\theta) d\bar{x} \\
&= \mathbb{E}_{\theta} [f(\bar{x}) \nabla_{\theta} \log \pi(\bar{x}|\theta)]
\end{aligned} \tag{7.19}$$

In order to conduct the mapping into an exponential, natural space, Glasmachers et al. introduce a change of variables described in Equation (7.20) such that \mathcal{N} ($\delta = 0, M = 0$) is equivalent to \mathcal{N} (μ, A) [191].

$$\begin{aligned}
\delta &\rightarrow \mu_{new} = \mu + A\delta \\
M &\rightarrow A_{new} = A \exp\left(\frac{1}{2}M\right) \\
\mathcal{N}(\mu, A) &= \mathcal{N}(\delta = 0, M = 0)
\end{aligned} \tag{7.20}$$

It can be seen that the original variables (μ, A) are recovered when δ and M go to zero when looking at Equation 7.20. Using this change of variables, the new search gradients as well as their approximation using unscented sample points, $\bar{\chi}_i$, and weights, $W_{\sigma,i}$, are defined in Equation (7.21). Where $J(0, 0)$ indicates that they are evaluated at $\delta = 0$ and $M = 0$ to recover the original variables μ and A .

$$\begin{aligned}
\nabla_{\delta} J(0, 0) &= \mathbb{E}_{\delta} [f(\bar{x}) \nabla_{\delta}|_{\delta=0} \log \pi(\bar{x}|\delta, M = 0)] \\
&\approx \sum_{i=1}^{\lambda} f(\bar{\chi}_i) W_{\sigma,i} \nabla_{\delta}|_{\delta=0} \log \pi(\bar{\chi}_i|\delta, M = 0) \\
\nabla_M J(0, 0) &= \mathbb{E}_M [f(\bar{x}) \nabla_M|_{M=0} \log \pi(\bar{x}|\delta = 0, M)] \\
&\approx \sum_{i=1}^{\lambda} f(\bar{\chi}_i) W_{\sigma,i} \nabla_M|_{M=0} \log \pi(\bar{\chi}_i|\delta = 0, M)
\end{aligned} \tag{7.21}$$

The next step is to define the various terms in the search gradients, depicted in Equation (7.21), beginning with the density function evaluated and natural log of the density function evaluated at the sigma points $\bar{\chi}$. Before applying the exponential, natural variables δ and M , these functions are represented in the form shown in Equation (7.22). This form uses a factorization of the covariance matrix, A where $AA^T = C$ so that $\theta = [\mu, A]$.

$$\begin{aligned}\pi(\bar{\chi}|\theta) &= \frac{1}{(\sqrt{2\pi})^n |det(A)|} \cdot exp\left(-\frac{1}{2}\|A^{-1} \cdot (\bar{\chi} - \mu)\|^2\right) \\ log\pi(\bar{\chi}|\theta) &= -\frac{n}{2} \log(2\pi) - \log(det(A)) - \frac{1}{2}\|A^{-1} \cdot (\bar{\chi} - \mu)\|^2\end{aligned}\quad (7.22)$$

From here, the next step is to conduct the mapping of the log density function in Equation (7.22) to the natural, exponential coordinate system using the new exponential variables outlined in Equation (7.20). A simple substitution of variables leads to the new form of the log density function outlined as follows.

$$\begin{aligned}log\pi(\bar{\chi}|\delta, M) &= -\frac{n}{2} \log(2\pi) - \log\left(det\left(A \exp\left(\frac{1}{2}M\right)\right)\right) \\ &\quad - \frac{1}{2}\left\|\left(A \exp\left(\frac{1}{2}M\right)\right)^{-1} \cdot (\bar{\chi} - (\mu + A\delta))\right\|^2\end{aligned}$$

Next, this logarithmic density function undergoes several simplification steps to get it into a form that is easier to work with when calculating its gradients. First of all, given an arbitrary matrix D the property $\exp(D)^{-1} = \exp(-D)$ is used to produce the following:

$$\begin{aligned}log\pi(\bar{\chi}|\delta, M) &= -\frac{n}{2} \log(2\pi) - \log\left(det\left(A \exp\left(\frac{1}{2}M\right)\right)\right) \\ &\quad - \frac{1}{2}\left\|\exp\left(-\frac{1}{2}M\right)A^{-1} \cdot (\bar{\chi} - (\mu + A\delta))\right\|^2\end{aligned}$$

Next, given two arbitrary matrices D and E the property $\det(DE) = \det(D)\det(E)$ is applied to this log density function.

$$\begin{aligned} \log\pi(\bar{\chi}|\delta, M) &= -\frac{n}{2}\log(2\pi) - \log\left(\det(A)\det\left(\exp\left(\frac{1}{2}M\right)\right)\right) \\ &\quad - \frac{1}{2}\left\|\exp\left(-\frac{1}{2}M\right)A^{-1} \cdot (\bar{\chi} - (\mu + A\delta))\right\|^2 \end{aligned}$$

From here, the matrix relationship $\det(\exp(D)) = \exp(\text{tr}(D))$ is applied to the Gaussian log density function to produce the following:

$$\begin{aligned} \log\pi(\bar{\chi}|\delta, M) &= -\frac{n}{2}\log(2\pi) - \log(\det(A)) - \log\left(\exp\left(\text{tr}\left(\frac{1}{2}M\right)\right)\right) \\ &\quad - \frac{1}{2}\left\|\exp\left(-\frac{1}{2}M\right)A^{-1} \cdot (\bar{\chi} - (\mu + A\delta))\right\|^2 \end{aligned}$$

The last simplification step uses the property $\log(\exp(D)) = D$ resulting in the mapped log density function of a Gaussian outlined in Equation (7.23).

$$\begin{aligned} \log\pi(\bar{\chi}|\delta, M) &= -\frac{n}{2}\log(2\pi) - \log(\det(A)) - \text{tr}\left(\frac{1}{2}M\right) \\ &\quad - \frac{1}{2}\left\|\exp\left(-\frac{1}{2}M\right)A^{-1} \cdot (\bar{\chi} - (\mu + A\delta))\right\|^2 \end{aligned} \tag{7.23}$$

Now that the Gaussian log density function has successfully been mapped to the exponential, natural space, the gradients of the Gaussian log density function with respect to the exponential, natural variables δ and M are needed to calculate the mapped expectation outlined in Equation (7.21). First, the new gradient of the Gaussian log density function with respect to δ , $\nabla_{\delta}|_{\delta=0} \log\pi(\bar{\chi}|\delta, M=0)$, is developed in Equation (7.24). Starting with Equation (7.23) and letting $M = 0$ produces the following:

$$\log\pi(\bar{\chi}|\delta, M=0) = -\frac{n}{2}\log(2\pi) - \log(\det(A)) - \frac{1}{2}\left\|A^{-1} \cdot (\bar{\chi} - (\mu + A\delta))\right\|^2$$

Next, the definition of an L2 norm is used to further simplify this expression. Terms that are independent of δ have been removed for simplification.

$$\begin{aligned}\nabla_{\delta}|_{\delta=0} \log \pi(\bar{\chi}|\delta, M=0) &= \nabla_{\delta}|_{\delta=0} \left[-\frac{1}{2} \left(\sqrt{\left(A^{-1} (\bar{\chi} - (\mu + A\delta)) \right)^2} \right)^2 \right] \\ &= \nabla_{\delta}|_{\delta=0} \left[-\frac{1}{2} \left(A^{-1} (\bar{\chi} - (\mu + A\delta)) \right)^2 \right]\end{aligned}$$

From this point, the chain rule is applied to take the gradient of this function w.r.t. δ producing the following result:

$$\nabla_{\delta}|_{\delta=0} \log \pi(\bar{\chi}|\delta, M=0) = \left[- \left(A^{-1} (\bar{\chi} - \mu - A\delta) \right) (-AA^{-1})^T \right]_{\delta=0}$$

After evaluating this expression at $\delta = 0$ the function is further simplified.

$$\nabla_{\delta}|_{\delta=0} \log \pi(\bar{\chi}|\delta, M=0) = A^{-1} (\bar{\chi} - \mu)$$

Next, the definition of $\bar{\chi} = \mu + A\chi$ is used and substituted into this expression to generate the following:

$$\nabla_{\delta}|_{\delta=0} \log \pi(\bar{\chi}|\delta, M=0) = A^{-1} (\mu + A\chi - \mu)$$

Lastly, Equation (7.24) results from further simplifying terms to produce a simple expression for the gradient of the Gaussian log density function w.r.t. δ evaluated at $(\delta = 0, M = 0)$.

$$\nabla_{\delta}|_{\delta=0} \log \pi(\bar{\chi}|\delta, M=0) = \chi \tag{7.24}$$

Now that $\nabla_{\delta}|_{\delta=0} \log \pi(\bar{\chi}|\delta, M=0)$ has been derived, the gradient of the expected fitness with respect to M , $\nabla_M|_{M=0} \log \pi(\bar{\chi}|\delta=0, M)$, is solved for using a similar approach. Starting with Equation (7.23) and letting $\delta = 0$ produces the following:

$$\log \pi(\bar{\chi}|\delta=0, M) = -\frac{n}{2} \log(2\pi) - \log(\det(A)) - \text{tr} \left(\frac{1}{2} M \right) - \frac{1}{2} \left\| \exp \left(-\frac{1}{2} M \right) A^{-1} \cdot (\bar{\chi} - \mu) \right\|^2$$

As was done previously, the definition of the L2 norm is utilized to produce the following expression for the gradient of the log density function w.r.t. M . Terms that are independent

of M have been removed for simplification.

$$\nabla_M|_{M=0} \log \pi(\bar{\chi}|\delta=0, M) = \nabla_M|_{M=0} \left[-\text{tr} \left(\frac{1}{2} M \right) - \frac{1}{2} \left(\exp \left(-\frac{1}{2} M \right) A^{-1} (\bar{\chi} - \mu) \right)^T \right]$$

Next, the chain rule is applied to find the gradient of this expression w.r.t. M . In addition, the property $\frac{\partial}{\partial D} \text{tr}(D) = I$ is used for finding the gradient of the trace of a positive definite matrix.

$$\nabla_M|_{M=0} \log \pi(\bar{\chi}|\delta=0, M) = \left[-I \frac{1}{2} - \left[\exp \left(-\frac{1}{2} M \right) A^{-1} (\bar{\chi} - \mu) \right] \left(-\frac{1}{2} I \exp \left(-\frac{1}{2} M \right) [A^{-1} (\bar{\chi} - \mu)]^T \right) \right]_{M=0}$$

At this point, this function can be evaluated at $M = 0$ to produce the following result:

$$\nabla_M|_{M=0} \log \pi(\bar{\chi}|\delta=0, M) = \left[-I \frac{1}{2} - [A^{-1} (\bar{\chi} - \mu)] \left(-\frac{1}{2} I \right) [A^{-1} (\bar{\chi} - \mu)]^T \right]$$

Next, the definition of $\bar{\chi}$ is again used such that the substitution $\bar{\chi} = \mu + A\chi$ can be made.

$$\nabla_M|_{M=0} \log \pi(\bar{\chi}|\delta=0, M) = \left[-I \frac{1}{2} - [A^{-1} (\mu + A\chi - \mu)] \left(-\frac{1}{2} I \right) [A^{-1} (\mu + A\chi - \mu)]^T \right]$$

Finally, this expression can be further simplified to produce Equation (7.25).

$$\nabla_M|_{M=0} \log \pi(\bar{\chi}|\delta=0, M) = \frac{1}{2} (\chi \chi^T - I) \quad (7.25)$$

Given that both $\nabla_\delta|_{\delta=0} \log \pi(\bar{\chi}|\delta, M = 0)$ and $\nabla_M|_{M=0} \log \pi(\bar{\chi}|\delta=0, M)$ have now been solved for, as outlined in Equation (7.24) and Equation (7.25), all of the necessary terms for the new search gradients have been found. The new search gradient estimates resulting from the xNES mapping and application of sigma points can be found by substituting $\nabla_\delta|_{\delta=0} \log \pi(\bar{\chi}|\delta, M = 0)$ and $\nabla_M|_{M=0} \log \pi(\bar{\chi}|\delta=0, M)$ back into Equation (7.21), as illustrated in Equation (7.26). This results in the new unscented form of the natural, exponential search gradient estimates. In this new coordinate frame, the Fischer information matrices become the identity matrix, resulting in a further simplified algorithm that no longer

needs to calculate these matrices or invert them.

$$\begin{aligned}\nabla_{\delta} J(0, 0) &\approx \sum_{i=1}^{\lambda} f(\bar{\chi}_i) W_{\sigma,i} \chi_i \\ \nabla_M J(0, 0) &\approx \frac{1}{2} \sum_{i=1}^{\lambda} f(\bar{\chi}_i) W_{\sigma,i} (\chi_i \chi_i^T - I)\end{aligned}\tag{7.26}$$

Next, by introducing learning rates η_{μ} and η_A , the parameter updates for the uxNES can be calculated with sigma points and weights as depicted in Equation (7.27). This exercise has shown that these equations are directly derived from the original expectation in Equation (7.19). It is important to verify this and confirm that the choice of both sigma point and probabilistic Gauss-Hermite discretization techniques are appropriate to enhance the performance of the xNES by more efficiently solving this Gaussian expectation. This is indeed the case, given that the Gaussian expectation is still at the center of the search gradient calculations and has simply been mapped to a new coordinate frame without altering the fundamental form of the equation.

$$\begin{aligned}\mu^{(g+1)} &= \mu^{(g)} + \eta_{\mu} \sum_{i=1}^{\lambda} f(\bar{\chi}_i) W_{\sigma,i} \chi_i \\ A^{(g+1)} &= A^{(g)} \exp\left(\frac{\eta_A}{2} \sum_{i=1}^{\lambda} f(\bar{\chi}_i) W_{\sigma,i} (\chi_i \chi_i^T - I)\right)\end{aligned}\tag{7.27}$$

Algorithm 4 illustrates how the uxNES works by incorporating all of the steps to include these new updates for the mean and Cholesky factor of the covariance matrix.

Algorithm 4: uxNES Algorithm (Version 1)

Initialize: $\mu^{(0)} = \text{random uniform} \in [LB, UB]^{\mathbb{R}^n}$;

User-Defined Parameters: g_{max} , η_μ , η_A , and $A^{(0)}$;

$g = 0$;

while stopping criteria not satisfied **do**

 Transform unscented points for given μ and A ;

$\bar{\chi}_k = \mu^{(g)} + A^{(g)} \chi_k$;

 Evaluate $f(\bar{\chi}_k)$ $k = [1, \dots, \lambda]$;

 Determine search gradients;

$\nabla_\delta J(0, 0) \approx \sum_{i=1}^{\lambda} f(\bar{\chi}_i) W_{\sigma,i} \chi_i$;

$\nabla_M J(0, 0) \approx \frac{1}{2} \sum_{i=1}^{\lambda} f(\bar{\chi}_i) W_{\sigma,i} (\chi_i \chi_i^T - I)$;

 Update parameters μ and A ;

$\mu^{(g+1)} = \mu^{(g)} + \eta_\mu \nabla_\delta J(0, 0)$;

$A^{(g+1)} = A^{(g)} \exp(\eta_A \nabla_M J(0, 0))$;

$g = g + 1$;

end

Lastly, Glasmachers et al. add a degree of flexibility to the algorithm by further factoring the A matrix into a σ_x and B component [191]. Using this factorization, σ_x is a scalar and B is a matrix where $\sigma_x B = A$. Using this method, the parameter updates are described in Equation (7.28) where η_σ and η_B are the learning rates that correspond to these factored components of the A matrix. The exponential form of the NES is utilized for the remainder of this chapter due to its increased robustness in that it ensures all covariance updates produce feasible, positive definite covariance values. In addition, it runs faster as it no longer requires the calculation and inversion of the Fischer information matrix as the standard NES and uNES did [191].

$$\begin{aligned}
\mu^{(g+1)} &= \mu^{(g)} + \eta_\mu \sum_{i=1}^{\lambda} f(\bar{\chi}_i) W_{\sigma,i} \chi_i \\
\sigma_x^{(g+1)} &= \sigma_x^{(g)} \exp \left(\frac{\eta_\sigma}{2} \frac{\operatorname{tr} \left(\sum_{i=1}^{\lambda} f(\bar{\chi}_i) W_{\sigma,i} (\chi_i \chi_i^T - I) \right)}{n} \right) \\
B^{(g+1)} &= B^{(g)} \exp \left(\frac{\eta_B}{2} \left(\sum_{i=1}^{\lambda} f(\bar{\chi}_i) W_{\sigma,i} (\chi_i \chi_i^T - I) - \frac{\operatorname{tr} \left(\sum_{i=1}^{\lambda} f(\bar{\chi}_i) W_{\sigma,i} (\chi_i \chi_i^T - I) \right)}{n} \cdot I \right) \right)
\end{aligned} \tag{7.28}$$

Algorithm 5 illustrates the full algorithm using the factored $A = \sigma B$ matrix approach for additional control and flexibility in adapting the distribution covariance. It can be observed that the exact same gradients $\nabla_\delta J(0, 0)$ and $\nabla_M J(0, 0)$ are still identical to what was solved for in the previous exercise, but now the A matrix update is simply factored into translational and rotational components. This is the form of the update equations that are used for the uxNES in the following sections; however, the covariance learning rates η_σ and η_B are set

equal to each other which is equivalent to using the form depicted in Algorithm 4.

Algorithm 5: uxNES Algorithm (Version 2)

Initialize: $\mu^{(0)}$ = random uniform $\in [LB, UB]^{\mathbb{R}^n}$;
User-Defined Parameters: g_{max} , η_μ , η_σ , η_B , and $A^{(0)}$;
 $\sigma_x^{(0)} = |det(A^{(0)})|^{1/n}$;
 $B^{(0)} = \frac{A^{(0)}}{\sigma_x^{(0)}}$;
 $g = 0$;
while stopping criteria not satisfied **do**
 Transform unscented points for given μ , σ_x , and B ;
 $\bar{\chi}_k = \mu^{(g)} + \sigma_x^{(g)} B^{(g)} \chi_k$;
 Evaluate $f(\bar{\chi}_k)$ $k = [1, \dots, \lambda]$;
 Determine search gradients;
 $\nabla_\delta J(0, 0) \approx \sum_{i=1}^\lambda f(\bar{\chi}_i) W_{\sigma,i} \chi_i$;
 $\nabla_M J(0, 0) \approx \frac{1}{2} \sum_{i=1}^\lambda f(\bar{\chi}_i) W_{\sigma,i} (\chi_i \chi_i^T - I)$;
 $\nabla_{\sigma_x} J(0, 0) \approx \frac{tr(\nabla_M J(0, 0))}{n}$;
 $\nabla_B J(0, 0) \approx \nabla_M J(0, 0) - \nabla_{\sigma_x} J(0, 0) I$;
 Update parameters μ , σ_x , and B ;
 $\mu^{(g+1)} = \mu^{(g)} + \eta_\mu \sigma_x B \nabla_\delta J(0, 0)$;
 $\sigma_x^{(g+1)} = \sigma_x^{(g)} \exp(\eta_\sigma \nabla_{\sigma_x} J(0, 0))$;
 $B^{(g+1)} = B^{(g)} \exp(\eta_B \nabla_B J(0, 0))$;
 $g = g + 1$;
end

7.6 Performance of uxNES

With the development of the uxNES as an algorithm that incorporates 3rd order sigma points to efficiently and accurately calculate search gradients, the next step is to apply this algorithm to the set of benchmark problems outlined in Table 6.1. Initial results on these problems reveal a fitness scaling issue that has to be addressed. A dynamic scaling technique is introduced to further increase the robustness of the uxNES over a range of potential magnitudes of objective function values. In addition, the problem-dependent nature of the covariance learning rates as compared to algorithm performance is investigated

so that the appropriate rates can be chosen for the various benchmark test problems. Lastly, this section compares the standard xNES and uxNES on 10-D, 20-D, and 40-D versions of the benchmark test functions.

A simple analysis on a 10-D parabolic function illustrates the improvements in convergence speed and accuracy for the function, $f(\bar{x}) = \sum_{i=1}^{10} \bar{x}_i^2$. Figure 7.8 and Table 7.12 illustrate these results. It can be seen that the standard xNES results in a fitness progression that is a little noisy as a result of random sampling whereas the fitness progression is smooth for the uxNES. It is also interesting to note that performance does not significantly improve with the addition of more random sample points. For this reason, the solution quality studies where the number of sample points are incrementally increased until solution quality is matched are not conducted in this chapter, however simple side-by-side average fitness comparisons are still performed. The reason that the results do not significantly improve by increasing the number of sample points is due to the incorporation of fitness-shaping into the search gradient calculation.

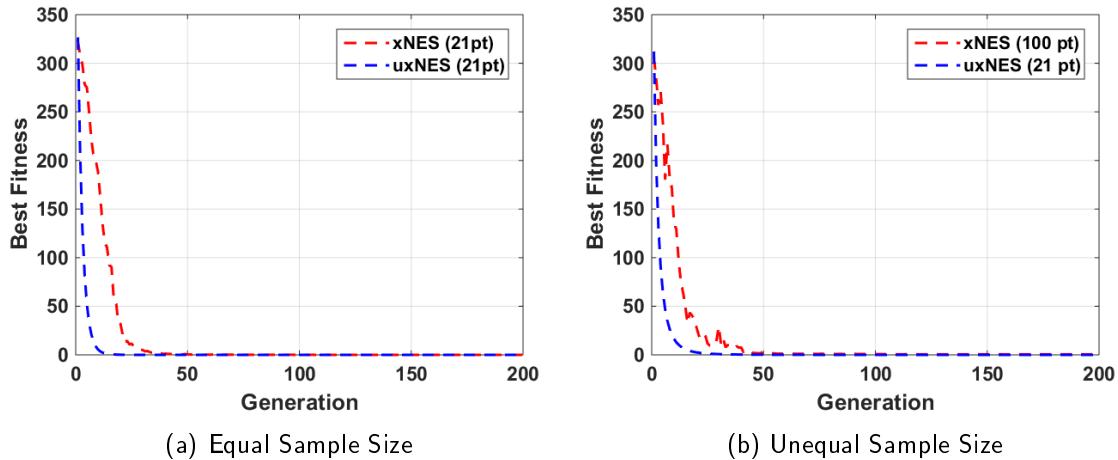


Figure 7.8. uxNES/xNES comparison on a 10-D parabolic function

Algorithm	$f(\bar{x})$ Evals	$f(\bar{x}_{best})$	C_0	$\eta_\sigma = \eta_B$	η_μ
xNES (21 points)	4,200	0.003054	$0.1 \times I$	$\frac{3(3+\log(n))}{5n\sqrt{n}}$	1.0
xNES (100 points)	20,000	0.52205	$0.1 \times I$	$\frac{3(3+\log(n))}{5n\sqrt{n}}$	1.0
uxNES (21 points)	4,200	1.2×10^{-18}	$0.1 \times I$	$\frac{3(3+\log(n))}{5n\sqrt{n}}$	1.0

Table 7.12. uxNES/xNES comparison

A quick comparison of search gradient accuracy between using fitness shaping and not using fitness shaping further illustrates that significant improvement in accuracy is not obtained by increasing the number of sample points when using fitness shaping, as seen in Figure 7.9. This plot illustrates the average L2 error norm of $\nabla_\mu J(\theta)$ with a 10-D objective function $f(\bar{x}) = \sum_{i=1}^{10} \bar{x}_i^2$ where $C = I$ and the mean is set to zero. It can be seen that there is an optimal number of points to use with fitness shaped weights, and beyond that value, the accuracy begins to decrease. It should also be noted that this is plotted on a log-log scale such that the 3rd order sigma points have zero error and therefore do not show up on the plot.

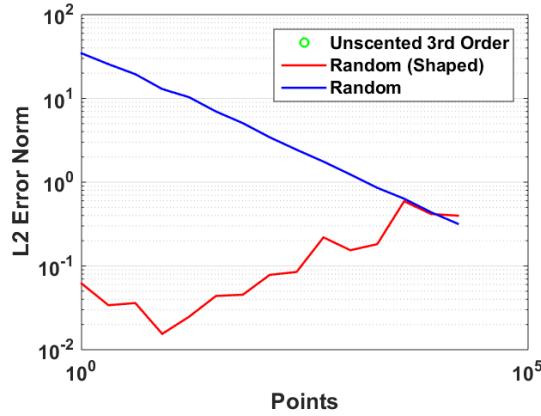


Figure 7.9. Estimate of $\nabla_\mu J(\theta)$ with 10-D parabolic function using fitness shaping

7.6.1 Dynamic Scaling

After initially applying the uxNES to the benchmark test problems, it is quickly observed that the algorithm is sensitive to the magnitude of the fitness values associated with the

various benchmark test problems. More specifically, several of the problems that had the potential to result in fairly large objective function values resulted in matrix scaling issues associated with MATLAB’s built-in matrix exponential function “`expm`.” In order to remedy this issue, a dynamic scaling technique is developed and incorporated into the uxNES. This scaling technique outlined in Equation (7.29) scales all of the fitness values so that they are within a reasonable range that works nicely with the matrix exponential function. In this technique, b is a user-defined factor that can further adjust the range of values. As an example, if $b = 1$ the range of possible scaled fitness values will be $[-1, 1]$. This process of scaling the fitness values is done during each generation, given that the value of $|F|_{max}$ will change from one generation to the next. This proves to be an effective method for automatically scaling the fitness values in such a way that the algorithm avoids scaling issues when it is applied to all of the benchmark test functions. It must be noted that the standard xNES avoids these issues by using the normalized, logarithmic fitness-shaping weights as described in Section 5.13.3. Fitness-shaping avoids these issues, as it is a form of fitness scaling in and of itself. With normalized fitness shaping weights, the sum of the shaped fitness values for any given set of offspring will always be equal to one.

$$\begin{aligned}\hat{f}(\bar{\chi}_k) &= \frac{f(\bar{\chi}_k)}{|F|_{max}} b \quad k = [1, \dots, \lambda] \\ |F|_{max} &= \max \{|f(\bar{\chi}_1)|, \dots, |f(\bar{\chi}_\lambda)|\}\end{aligned}\tag{7.29}$$

The updated uxNES with dynamic scaling is summarized in Algorithm 6 for completeness.

Algorithm 6: uxNES Algorithm (Version 2 With Scaling)

Initialize: $\mu^{(0)}$ = random uniform $\in [LB, UB]^{\mathbb{R}^n}$;
User-Defined Parameters: g_{max} , η_μ , η_σ , η_B , b , and $A^{(0)}$;
 $\sigma_x^{(0)} = |\det(A^{(0)})|^{1/n}$;
 $B^{(0)} = \frac{A^{(0)}}{\sigma_x^{(0)}}$;
 $g = 0$;
while stopping criteria not satisfied **do**
 Transform unscented points for given μ , σ_x , and B ;
 $\bar{\chi}_k = \mu^{(g)} + \sigma_x^{(g)} B^{(g)} \chi_k$;
 Evaluate $f(\bar{\chi}_k)$ $k = [1, \dots, \lambda]$;
 Scale fitness values;
 $\hat{f}(\bar{\chi}_k) = \frac{f(\bar{\chi}_k)}{|F|_{max}} b$ $k = [1, \dots, \lambda]$;
 Where $|F|_{max} = \max \{|f(\bar{\chi}_1)|, \dots, |f(\bar{\chi}_\lambda)|\}$;
 Determine search gradients;
 $\nabla_\delta J(0, 0) \approx \sum_{i=1}^\lambda \hat{f}(\bar{\chi}_i) W_{\sigma,i} \chi_i$;
 $\nabla_M J(0, 0) \approx \frac{1}{2} \sum_{i=1}^\lambda \hat{f}(\bar{\chi}_i) W_{\sigma,i} (\chi_i \chi_i^T - I)$;
 $\nabla_{\sigma_x} J(0, 0) \approx \frac{\text{tr}(\nabla_M J(0, 0))}{n}$;
 $\nabla_B J(0, 0) \approx \nabla_M J(0, 0) - \nabla_{\sigma_x} J(0, 0) I$;
 Update parameters μ , σ_x , and B ;
 $\mu^{(g+1)} = \mu^{(g)} + \eta_\mu \sigma_x B \nabla_\delta J(0, 0)$;
 $\sigma_x^{(g+1)} = \sigma_x^{(g)} \exp(\eta_\sigma \nabla_{\sigma_x} J(0, 0))$;
 $B^{(g+1)} = B^{(g)} \exp(\eta_B \nabla_B J(0, 0))$;
 $g = g + 1$;
end

7.7 Learning Rates

This section investigates the effects of changing the covariance learning rates used in the uxNES on each of the benchmark test problems. These learning rates are problem dependent where a few general problem characteristics may be useful in helping to determine what learning rates are appropriate for that specific problem. This set of experiments investigates five different learning rates for covariance as they apply to 10-D, 20-D, and

40-D versions of the benchmark test functions described in Table 6.1. The methodology for these experiments is similar to previous experiments in that the average fitness progression is observed over 100 runs for each case where each run consists of 10,000 generations. It is determined, that the uxNES works best on highly multimodal problems by starting out with a large initial covariance where each diagonal term in the covariance matrix is set to 10 times the initialization range for each of the problems defined in Table 6.1 such that $C^{(0)} = 10$ (Init. Range) I . This allows the algorithm to find global search directions, without getting trapped into a locally optimal basin on many of the problems. Table 7.13 illustrates that while the learning rates do not have a significant impact on algorithm performance for many of the problems, they do have a fairly large impact on a couple of problems such as the Griewank and Schaffer problems. Referring back to the 3-D visualizations of these test functions (Figure 6.3), it can be observed that the Griewank problem does not have a prominent “global search direction” that trends toward the global optimal solution. In other words, there are a number of locally optimal points that are very close in fitness value to that of the globally optimal point. With this specific function, the slower covariance learning rates appear to perform better as they allow the algorithm more time to explore the search space and to determine the subtle, global search direction. The uxNES has the most difficulty with the Skewed Rastrigin and Schwefel functions, where the global search gradient is relatively flat. On these problems, there is a need to have a large initial covariance coupled with an initially slow learning rate. However, once the algorithm finds a good region of the search space, a faster learning rate would be desirable so that it can exploit the local basin. Unfortunately, with the fixed covariance learning rates used in the uxNES, the practitioner is forced to compromise on the rates and try to find one that allows for a little of both desired characteristics to occur. On the other hand, problems such as Ackley and Bohachevsky have a very pronounced global search direction, where the uxNES is able to quickly progress in the direction of the globally optimal solution over a range of learning rates.

Learning Rates ($\eta_B = \eta_\sigma$):	1.0	0.1	0.01	0.001	0.0001
Ackley 10-D	0	0	0	0	6.9×10^{-15}
Ackley 20-D	0	0	0	5.3×10^{-16}	7.1×10^{-15}
Ackley 40-D	0	0	0	7.1×10^{-15}	1.6×10^{-14}
Bohachevsky 10-D	0	0	0	0	0
Bohachevsky 20-D	0	0	0	0	0
Bohachevsky 40-D	0	0	0	0	0
Griewank 10-D	0.0211	0.0042	0.0025	2.7×10^{-7}	0
Griewank 20-D	0.0070	0.0088	3.6×10^{-4}	0	0
Griewank 40-D	8.9×10^{-5}	5.1×10^{-5}	0	0	0
Rastrigin 10-D	0.2048	0.8597	0.0682	2.2×10^{-12}	0
Rastrigin 20-D	0	0	0	0	0
Rastrigin 40-D	0	0	0	0	0
Scaled Rastrigin 10-D	0.2532	0.0120	0	0	0
Scaled Rastrigin 20-D	0	0	0	0	0
Scaled Rastrigin 40-D	0	0	0	0	0
Skewed Rastrigin 10-D	64.9734	120.8131	121.2914	288.5741	313.3798
Skewed Rastrigin 20-D	171.7372	151.8842	177.5451	322.4582	319.3608
Skewed Rastrigin 40-D	371.9	159.2	787.3	739.5	706.3
Schaffer 10-D	0.0039	8.0×10^{-23}	0.1705	10.3971	13.0727
Schaffer 20-D	3.1×10^{-20}	3.2×10^{-22}	3.3778	157.4554	189.7161
Schaffer 40-D	2.0×10^{-19}	3.4×10^{-21}	238.6	469.3	478.8
Schwefel 10-D	3,534	2,340	2,131	2,115	3,926
Schwefel 20-D	7,798	7,109	7,012	6,831	8,297
Schwefel 40-D	15,157	14,124	13,914	15,619	15,511

Table 7.13. uxNES average best fitness vs. covariance learning rates

It can be seen that the uxNES appears to be fairly robust in terms of varying the covariance rates on many of the problems. Figure 7.10 illustrates the average fitness progressions for the uxNES for five different covariance learning rates where $\eta_\sigma = \eta_B$. Table 7.13 indicates that the algorithm is able to obtain an average fitness value of zero, meaning that it finds the globally optimal solution during all of the 100 runs for a number of the problems. However, Figure 7.10a and c show more detail that allow for the determination of optimal learning rates in terms of the average convergence speed at which the algorithm finds the globally optimal point. It is interesting to note that on both the Bohachevsky and the Scaled Rastrigin functions, the optimal learning rates are not the fastest or the slowest rates, but the ones that are in the middle of the range of rates that are investigated. Figure 7.10b and d illustrate the cases where the learning rates had a more significant effect on algorithm

performance. On the Griewank function, the slowest covariance learning rates result in the best performance, while on the Schaffer function, the fastest rates produced the best average fitness values. While the 3-D visualizations of these problems may offer some insight into what covariance learning rates might work the best for each problem, this visualization of the fitness landscape is seldom available when working with real world problems and is still speculative at best.

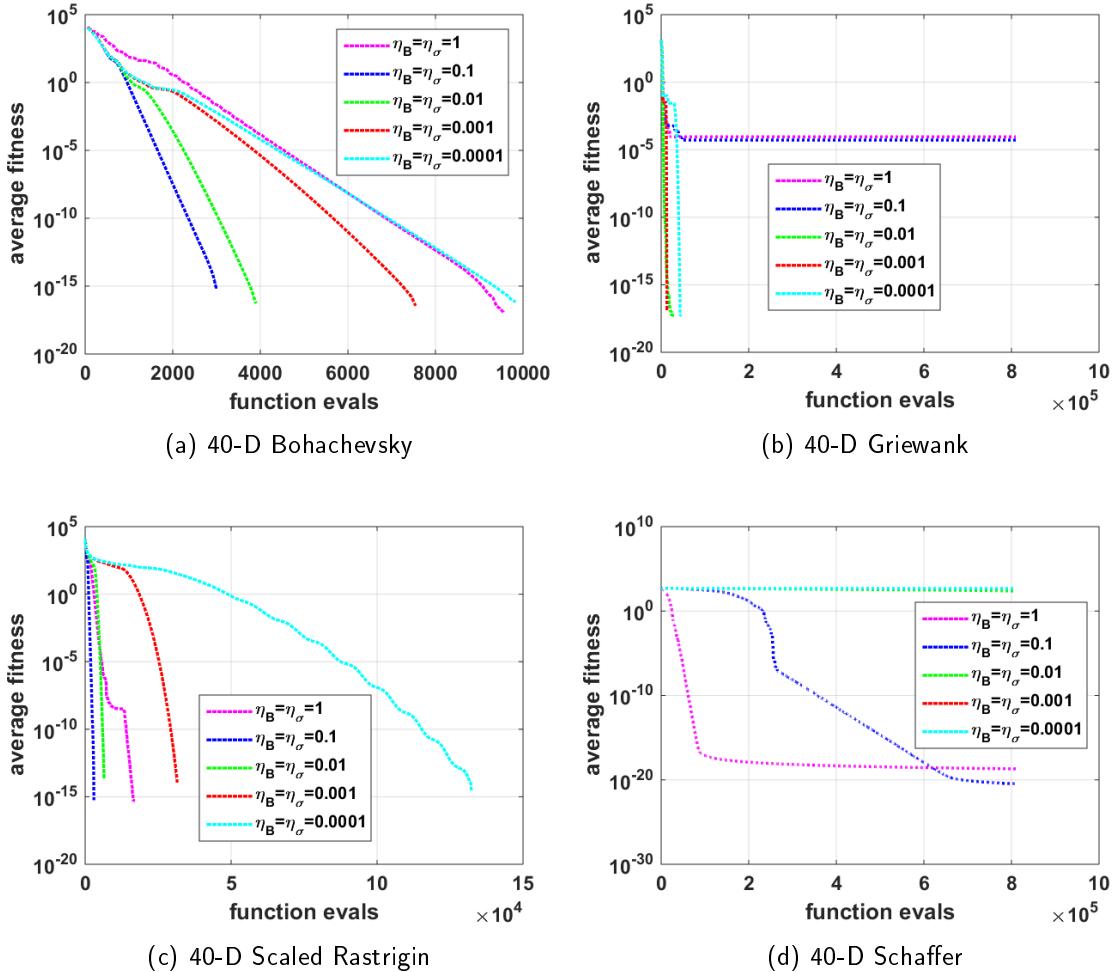


Figure 7.10. Average uxNES fitness progression on 40-D problems with different learning rates

Lastly, the affect of learning rates on the trace of the covariance matrix can be seen in Figure

7.11. For some of the faster learning rates, it can be seen that the covariance seems to stagnate or converge to a certain covariance that stops changing as the generations progress. In the Griewank problem, these faster covariance rates led to premature convergence where the algorithm does not always find the globally optimal solution resulting in an average fitness progression that does not trend to zero.

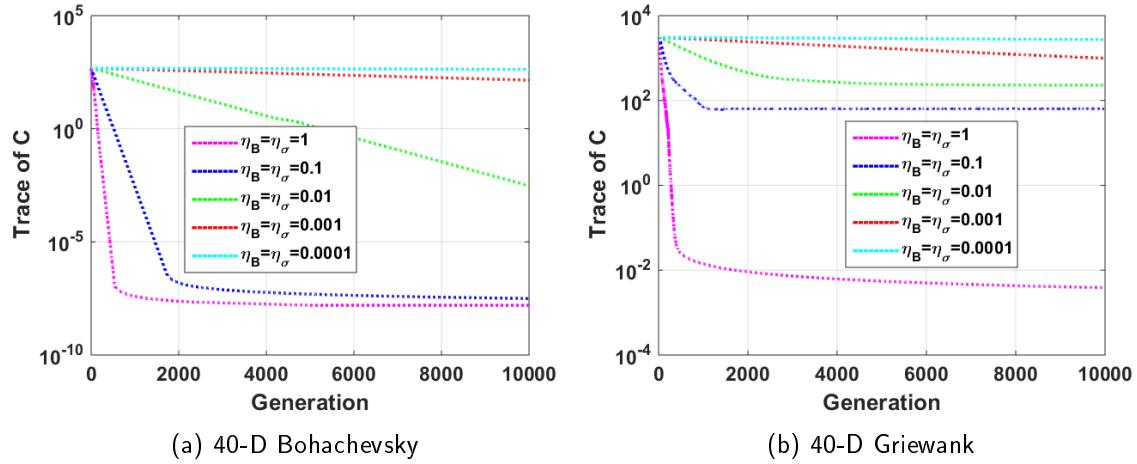


Figure 7.11. Average uxNES $tr(C)$ progression on 40-D problems with different learning rates

7.8 xNES/uxNES/USA-ES Comparison on Test Functions

This section compares the performance of the xNES, uxNES, and USA-ES on the benchmark test problems outlined in Table 6.1. An open source version of the xNES that was written by Sun Yi is utilized for these comparison studies. This version of the xNES does not factor the A matrix into multiple components and only uses a single covariance learning rate $\eta_A = 0.5\min\left\{\frac{1}{n}, 0.25\right\}$. In addition, the USA-ES is also applied to these problems with the same covariance decay method that is described in Equation (6.1). The learning rates from Table 7.13 are utilized for the uxNES, as it seems to perform better with slower covariance learning rates that prevent premature convergence. These slower learning rates did not benefit the performance of the standard xNES as they did in the uxNES. It is for this reason that the xNES is applied to this set of problems with its standard initial parameter settings, as these settings produced the better average results across the set of problems in

the various dimensional cases when compared to running it with the same settings used for the uxNES. However, it must be noted that these settings may not be optimal for the standard xNES, and could likely be further tuned and improved upon for better results.

The methodology used for this set of experiments is very similar to previous ES comparison studies where an average fitness progression is observed for each algorithm. Given that some of these algorithms took a little longer to converge to final solutions than the algorithms considered in the previous chapter, each algorithm run consists of 5,000 generations (6,000 generations for the 40-D case). A uniformly distributed, random initial starting point is selected at the beginning of each run. This same starting point is used by each of the three algorithms for the given run so that all three algorithms start at the same randomly selected location in the search space. This process is repeated 100 times for each problem and the average fitness progression, as well as the standard deviation of these fitness progressions are determined for each algorithm variant on each of the problems. This data is then used to investigate the best average fitness values that each algorithm converges to, with a 99 % confidence interval to illustrate potential variation in results.

Function	xNES $f_{avg\ best}(\bar{x})$ 99% CI	uxNES $f_{avg\ best}(\bar{x})$ 99% CI	USA $f_{avg\ best}(\bar{x})$ 99% CI
Ackley	$3.6 \times 10^{-15} \pm 0.0$	0 ± 0	$3.2 \times 10^{-15} \pm 2.9 \times 10^{-16}$
Bohachevsky	0 ± 0	0 ± 0	0 ± 0
Griewank	0.306 ± 0.063	0 ± 0	0.021 ± 0.002
Rastrigin	6.268 ± 0.856	0 ± 0	0 ± 0
Scaled Rastrigin	6.965 ± 0.873	0 ± 0	0 ± 0
Skewed Rastrigin	13.631 ± 0.542	69.965 ± 3.117	7.761 ± 0.338
Schaffer	82.219 ± 1.362	$2.2 \times 10^{-22} \pm 0$	0 ± 0
Schwefel	$2,315.2 \pm 86.341$	$2,104.3 \pm 8.514$	736.332 ± 72.984

Table 7.14. 10-D xNES/uxNES/USA-ES results ($\lambda = 21$)

Function	xNES $f_{avg\ best}(\bar{x})$ 99% CI	uxNES $f_{avg\ best}(\bar{x})$ 99% CI	USA $f_{avg\ best}(\bar{x})$ 99% CI
Ackley	$3.6 \times 10^{-15} \pm 0$	0 ± 0	$3.6 \times 10^{-15} \pm 0$
Bohachevsky	0 ± 0	0 ± 0	0 ± 0
Griewank	0.010 ± 0.002	0 ± 0	0.022 ± 0.004
Rastrigin	9.154 ± 0.920	0 ± 0	0 ± 0
Scaled Rastrigin	11.044 ± 0.750	0 ± 0	0 ± 0
Skewed Rastrigin	20.894 ± 0.96	165.948 ± 5.30	8.159 ± 0.672
Schaffer	166.341 ± 2.0	$1.8 \times 10^{-21} \pm 1.0 \times 10^{-22}$	0.0017 ± 0.0014
Schwefel	$4,650.1 \pm 68.63$	$7,897.9 \pm 26.92$	607.991 ± 23.62

Table 7.15. 20-D xNES/uxNES/USA-ES results ($\lambda = 41$)

Function	xNES $f_{avg\ best}(\bar{x})$ 99% CI	uxNES $f_{avg\ best}(\bar{x})$ 99% CI	USA $f_{avg\ best}(\bar{x})$ 99% CI
Ackley	$9.7 \times 10^{-8} \pm 4.5 \times 10^{-9}$	0 ± 0	$3.6 \times 10^{-15} \pm 0$
Bohachevsky	$1.0 \times 10^{-12} \pm 3.5 \times 10^{-14}$	0 ± 0	0 ± 0
Griewank	$5.9 \times 10^{-15} \pm 3.3 \times 10^{-16}$	0 ± 0	0.017 ± 0.004
Rastrigin	15.025 ± 0.909	0 ± 0	0 ± 0
Scaled Rastrigin	12.345 ± 0.729	0 ± 0	0 ± 0
Skewed Rastrigin	40.097 ± 0.652	371.469 ± 1.537	0 ± 0
Schaffer	344.749 ± 2.333	$3.0 \times 10^{-15} \pm 1.1 \times 10^{-15}$	0.0178 ± 0.0051
Schwefel	$9,568.9 \pm 97.436$	$15,883.0 \pm 109.110$	$3.6 \times 10^{-12} \pm 0$

Table 7.16. 40-D xNES/uxNES/USA-ES results ($\lambda = 81$)

Tables 7.14-7.16 summarize the results obtained for the three various ESs used for comparison. It can be seen the uxNES is able to find the globally optimal solution for six of the eight problems in each dimensional case for all of the 100 trials. It must be pointed out that this new algorithm has difficulty with the Skewed Rastrigin and Schwefel functions. The standard xNES outperforms the uxNES on these particular problems, however, it still has difficulty locating the globally optimal point on these functions as well. The uxNES seems to perform the best on problems that have a more defined global search direction. Referring back to Figure 6.3, it can be seen that the fitness landscapes of the Schwefel and Skewed Rastrigin functions offer little insight in terms of global search direction. Whereas functions such as Ackley or Bohachevsky, while they have many locally optimal solutions, also have a fairly well defined global fitness trend that the uxNES can quickly find using search gradients. The simple USA-ES is able to outperform the other algorithms on both the Skewed Rastrigin and Schwefel functions, where it finds the globally optimal solution during all of the 100 trials for both of these problems in the 40-D case. It may seem counter-intuitive that the USA-ES performs better on these problems in the 40-D case than it does on the 20-D or 10-D cases. The reason for this is that the covariance decay rate is slower for the 40-D case based on Equation (6.1). This indicates that the USA-ES would likely do better in the lower dimensional cases, if the covariance decay rate were slightly reduced for these particular functions, again this is a user-defined setting that can be tuned. The USA-ES likely does better on these problems given that it does not rely as much on search gradient information, but more on the spread of its 3rd order sigma points. With this, the USA-ES is able to more effectively locate the globally optimal solution on these types of problems with little to no indication in terms of global search direction based on search gradients.

However, as depicted in Figure 7.12, the uxNES is very accurate and very fast on the majority of these multimodal problems where a global search direction can be determined. It's accurate search gradient determination using 3rd order sigma points enables the algorithm to quickly progress in the best global search direction to find globally optimal solutions. However, this improved search gradient accuracy does not address the issue of deceptive global search directions, or little to no distinguishable global trend in terms of improving/deteriorating fitness values. In these cases, the simplistic algorithms, such as the USA-ES, may have an advantage as is indicated by the results on the Skewed Rastrigin and Schwefel functions.

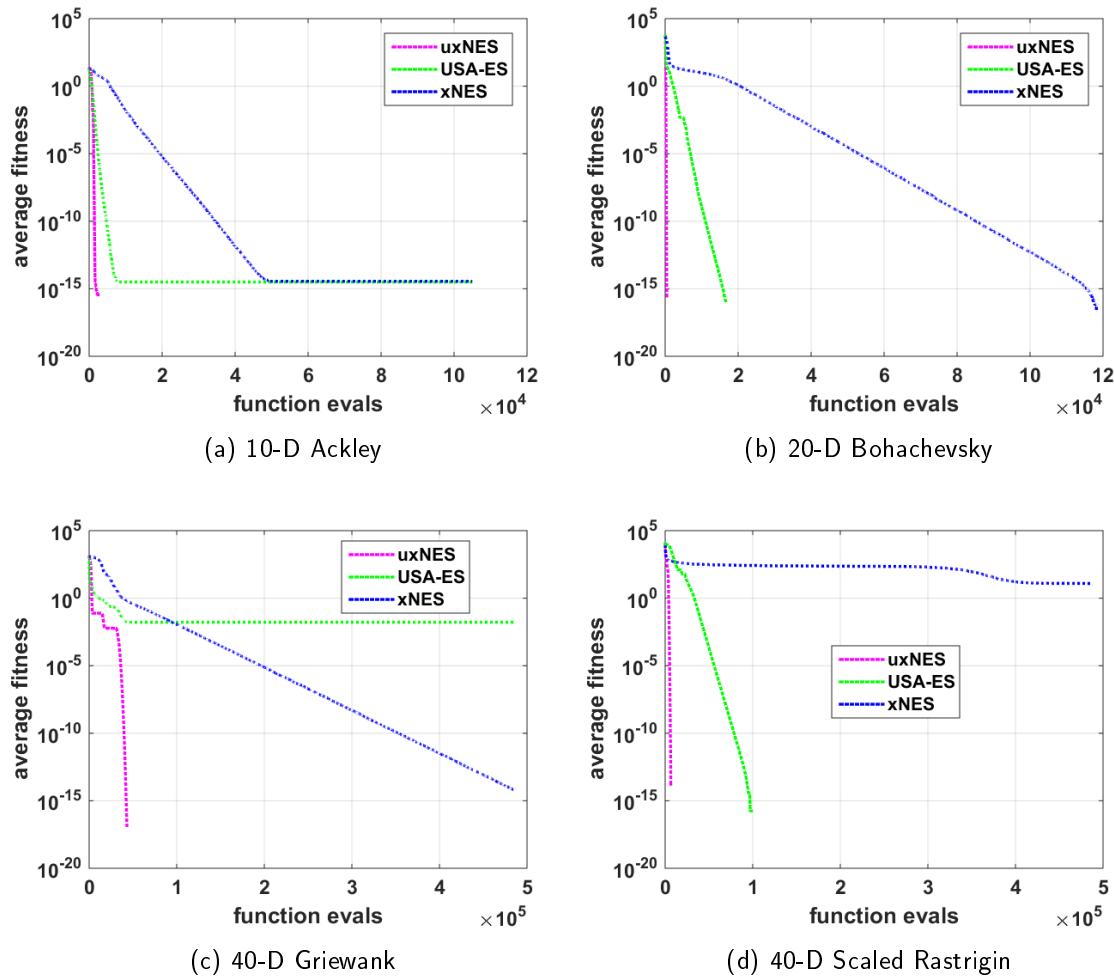


Figure 7.12. Average xNES/uxNES/USA-ES fitness progression on test problems

7.9 Global Optimization Algorithm Using uxNES

This section investigates a global optimization algorithm that utilizes the uxNES as a way to continuously improve upon a reigning optimal solution. This algorithm idea is initially introduced by and has been proven to converge as the number of iterations approaches infinity [205]. Algorithm 7 illustrates the general method used for this technique of iteratively improving upon a reigning optimal solution, $(f_{reigning}, \bar{x}_{reigning})$. The user must define several parameters that influence how this algorithm progresses. A penalty parameter, p , is needed for an exact penalty function that is used to penalize solutions that are worse than the reigning optimal. In addition, a growth rate, α , defines how the desired improvement, ϵ , in reigning optimal fitness changes between each uxNES run. The value ϵ can be dynamic if $\alpha > 1$, or static if $\alpha = 1$. When $\alpha > 1$ the value of ϵ is increased every time the uxNES is successful in finding a solution that is better than $(f_{reigning} - \epsilon)$ and ϵ is decreased every time the uxNES is unable to find a better solution after meeting its internal stopping criteria (e.g. maximum generation limit). In addition, the global optimization algorithm keeps track of how many failed attempts have occurred and uses this as an exit criteria. With this, a maximum number of attempts, $fail_{max}$, allowed for the optimizer to find a solution that is better than the reigning optimal by the desired amount, ϵ , is needed. Of course, a number

of other stopping criteria can also be used for this algorithm.

Algorithm 7: uxNES Global Optimization Algorithm

```

Initialize:  $\mu^{(0)}$ ,  $\epsilon$ ,  $f_{reigning}$ ,  $p$ ,  $\alpha$ , and  $fail_{max}$ ;
while  $Count < fail_{max}$  do
    Run uxNES;
    Penalize solutions using  $F(\bar{x}) = f(\bar{x}) + p \max \{0.0, f(\bar{x}) - (f_{reigning} - \epsilon)\}$ ;
    if solution found is better than  $(f_{reigning} - \epsilon)$  then
        break from uxNES;
        Update  $f_{reigning}$ ;
        Starting point,  $\mu^{(0)}$ , for next run set to  $\bar{x}_{reigning}$ ;
         $\epsilon = \epsilon \times \alpha$ ;
    else
        Run uxNES for max generations;
         $\epsilon = \frac{\epsilon}{\alpha}$ ;
         $Count = Count + 1$ ;
        Generate new  $\mu^{(0)}$  from uniform random distribution [LB,UB];
    end
end

```

An interesting characteristic of this algorithm is how it uses an exact penalty function to penalize all of the solutions that are not better than the reigning optimal solution. This has the effect of scaling areas of the search domain that are not producing better fitness values to potentially create steeper, more defined search gradients. This can make it easier for the uxNES to rapidly find regions of the search space that offer better solutions than the best solution found to date. In addition, after a local optimum is reached, the area will become penalized as it is no longer able to produce solutions that are better than the reigning optimal. The hope is that this will make it easier for the algorithm to escape these local optima. However, there is a chance that the globally optimal basin could become very isolated and small such that the problem is still difficult to solve, as it begins to look like a needle in a haystack problem.

In addition, upon a failed attempt to improve upon the reigning optimal solution, the algorithm randomly selects a new starting distribution mean location $\mu^{(0)}$ from a uniform

distribution, providing yet another mechanism that helps the uxNES escape local optima. Upon a successful run of the uxNES where the reigning optimal is improved, the starting point for the next run is set to this new reigning optimal solution, $\bar{x}_{reigning}$, so that the uxNES is able to pick up where it left off and fully exploit that particular region of the search space.

Figure 7.13 illustrates the results of applying this algorithm to the 10-D Skewed Rastrigin and Schwefel functions, as they are the two test problems that the uxNES has the most difficulty with. It can be seen that the final fitness found in both of these problems is significantly better than the average fitness values listed in Table 7.14. These runs utilized a dynamic ϵ where $\alpha = 2$. It can be seen in the plots that ϵ is increased during successful uxNES runs, and it is decreased for runs where the reigning optimal solution is not improved upon by more than the desired ϵ amount.

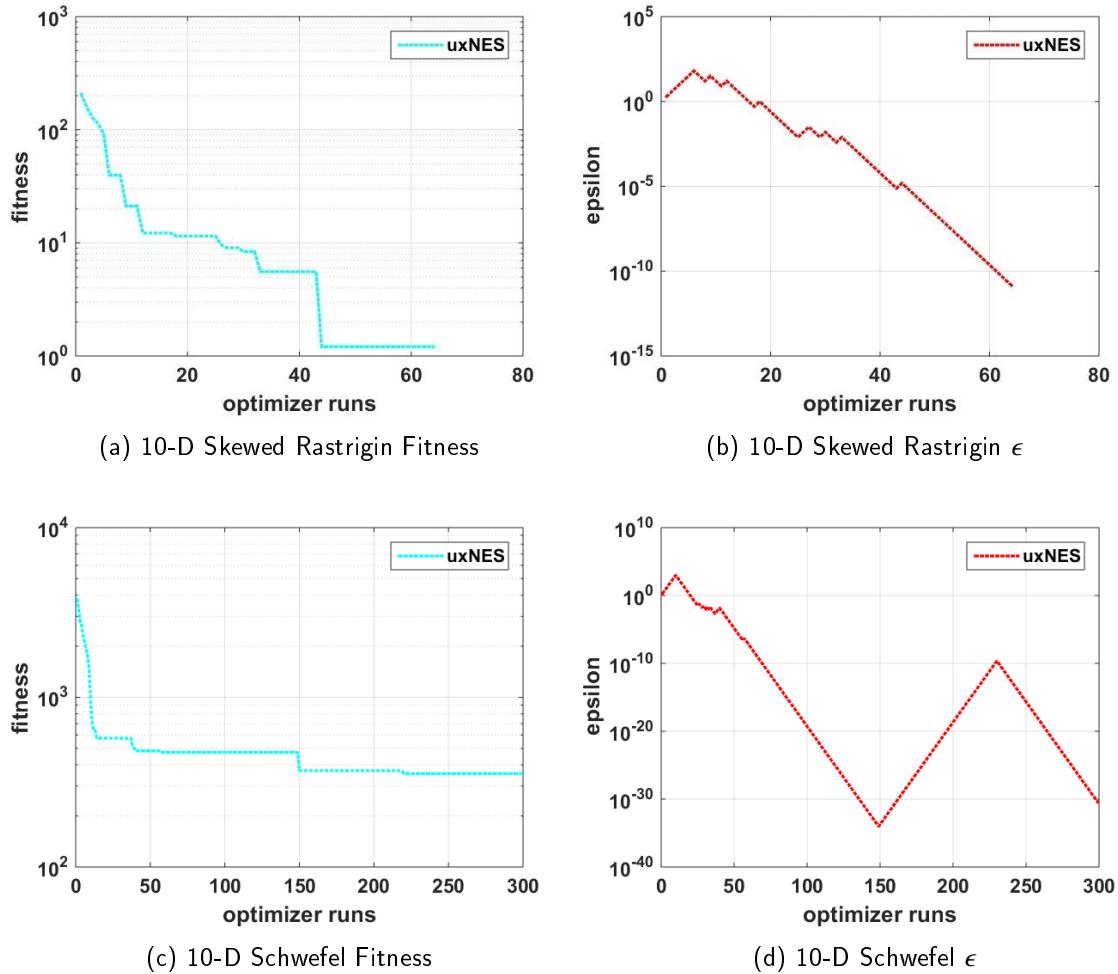


Figure 7.13. uxNES global optimization algorithm results for 10-D test functions

7.10 uxNES Lunar Lander Optimal Control Application

The uxNES has been developed and tested on the various test problems throughout this chapter. This section looks at how well this new type of ES is able to optimize the lunar lander problem introduced in Equation (4.3). As illustrated in Figure 7.14, the uxNES is able to quickly find a very good solution to this problem. It is able to arrive at nearly optimal solutions significantly faster than the various GAs and USA-ES, where all constraints and conditions are satisfied. The specific results are summarized in Table 7.17 where it can

be seen that this algorithm is able to find a feasible solution to the lunar lander problem that is within half of a percent of the known optimal solution. It is also impressive to see that this algorithm, using MATLAB code, is able to achieve this solution in under three minutes, which is very quick when compared to some of the other stochastic algorithms that have been investigated such as the GAs or USA-ES which were both coded in C. While the USA-ES found a slightly more accurate solution, it took approximately 10 times as long to get there. The trajectories illustrated in Figure 7.14 meet all of the constraints and end point conditions. It can be seen that the control trajectory is a little smoother than a discrete switching function that represents the true optimal trajectory. However, there are some advantages to having a smoother control trajectory when applying such a control to an actual system. As an example, real world systems are not likely able to produce instantaneous changes in thrust. With this, the smoother control curves produced by the uxNES may be more closely followed by real systems. In some cases, smoothing of control trajectories is desired and can be used to prevent saturation of the various control systems in real world applications.

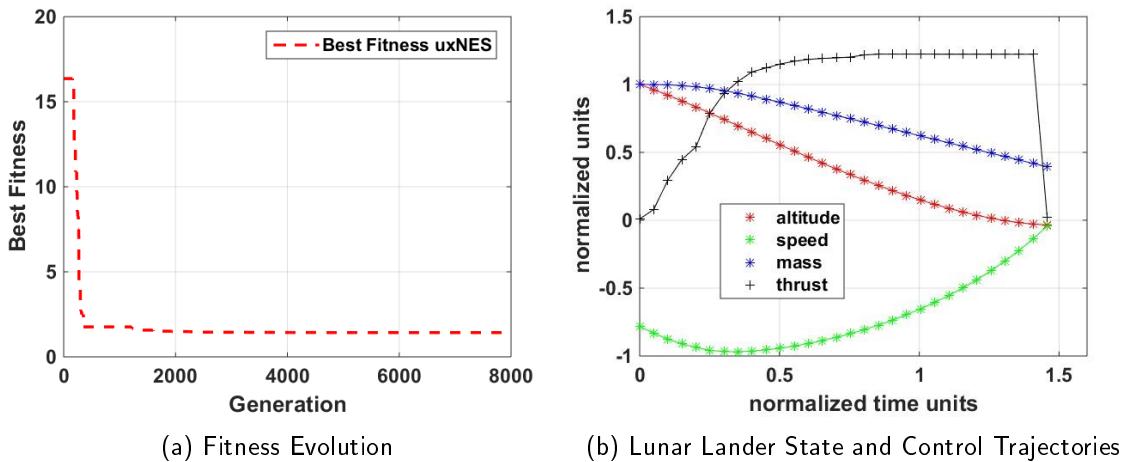


Figure 7.14. uxNES results on a 30-node lunar lander problem

$F(\bar{x})$	CPU Time (min)	Difference From Optimal (%)
1.4261	2.89	0.39

Table 7.17. uxNES results on a 30-node lunar lander problem

7.11 Conclusion

The idea of using deterministically chosen interpolation points within the construct of a NES is a unique and useful contribution toward these types of ESs. This chapter has taken the reader through the development and analysis of the uxNES, where 3rd order sigma points are utilized in conjunction with the exponential form of a NES to produce an accurate, and robust optimization algorithm. The more accurate search gradients produced by deterministic sampling and quadrature techniques are able to lead to faster convergence when compared to the use of random sampling. The uxNES is able to consistently find the globally optimal solution for six out of the eight multimodal test functions in all of the various dimensional cases. In addition, a global optimization algorithm is developed and shown to further enhance the performance of the uxNES on the more difficult test problems. Lastly, it is demonstrated that the uxNES proves to be an effective tool for solving the lunar lander optimal control problem. The uxNES appears to emulate some of the desirable traits of gradient-based search methods such as faster convergence times, while still having the ability to work with non-differentiable search domains and black box optimization problems. This is a very exciting and unique contribution that changes course from the standard methodology typically used with ESs. The uNES and uxNES are just the tip of the iceberg in terms of this new methodology of deterministically choosing sample points within the context of an ES. The hope is that this idea will help bridge the gap between stochastic and gradient-based optimization techniques in terms of speed and accuracy, while at the same time, still maintaining the desirable characteristics that stochastic methods have in terms of being able to work in discontinuous spaces, optimize multimodal problems, and solve black box optimization problems.

CHAPTER 8: Conclusions and Future Work

8.1 Introduction

While this chapter represents the culmination of this dissertation, it is hopefully the starting point for many more exciting research opportunities to come. The chapter begins by summarizing the unique contributions of this dissertation along with some of the challenges that they help address. It then discusses the future work relative to this research that could lead to further contributions and useful applications associated with astronautical engineering. Lastly, the chapter provides a discussion of several ideas for new and exciting areas of research that are closely associated with the techniques that have been explored throughout this dissertation.

8.2 Contributions and Relevancy

This section summarizes the unique contributions that are introduced and developed within this body of research as they were discovered in chronological order. The dissertation begins with a new application of parallel processing (Chapter 2) that can rapidly calculate RLV landing footprints. While the parallel computing aspects of this technique are not novel or unique, it is an application that has not been documented in literature and fully explored to this extent. Furthermore, the insights of appropriately balancing computational workload coupled with parallel processing memory architectures are useful in the context of this engineering optimal control problem. It must be noted that workload balancing and memory architecture studies have been done extensively, however, it is unique to study them through the lens of an RLV optimal control engineering application.

Chapter 4 conducts a unique trade study used to analyze the performance of exact penalty functions and inexact penalty functions from the perspective of parallel GAs and optimal control. It is discovered that the exact penalty functions, though not as frequently applied in GAs, seem to produce better results. Evolutionary algorithms do not require continuously differentiable functions and are able to work with exact penalty functions. The reason that

inexact penalty functions are often used in these algorithms is likely due to extensive work in the NLP community that has been done to find differentiable versions of penalty functions that can be used in gradient-based solvers. Given that these types of penalty functions were common practice when evolutionary algorithms came about, they were often used for these types of algorithms, though continuously differentiable functions were not needed.

Chapter 4 also introduces two new Gaussian based crossover operators: DTGX and DGX while conducting a unique study of parallel RCGAs as they apply to optimal control. While these new crossover operators didn't offer significant gains in performance when compared to state-of-the-art RCGA crossover operators, they did pave the way for the introduction of an unscented RCGA, outlined and developed in Appendix C. According to the literature survey conducted for this dissertation, this new form of deterministic sampling within the construct of a RCGA has never been documented before. This unscented RCGA proves to be a critical stepping stone for the development of the various ESs that utilize unscented sampling techniques. In addition, Appendix C, empirically shows that this new form of unscented GA is able to achieve very accurate results in solving a lunar lander optimal control problem when run on multiple processors in a parallel island architecture. It will be seen that this idea of an unscented RCGA is still very much in its infancy, and several new ideas are proposed later in Section 8.4 to further improve this algorithm.

The dissertation changes course in Chapter 6 and begins to investigate the idea of deterministic sampling within the construct of ESs. Initially, a fairly simple ES is developed by building upon the idea of the DGX crossover operator that was introduced in Chapter 4. The USA-ES is created by applying a DGX-inspired mutation operator and a simulated-annealing-inspired covariance adaptation scheme within the general construct of an ES. Empirical analysis shows that this new algorithm obtains enhanced performance in terms of accuracy and computational time when compared with a similar ES that utilizes Monte Carlo sampling. This new USA-ES proves to be very effective on many of the highly multimodal benchmark test problems as well as the lunar lander optimal control problem. Both 3rd order unscented sigma points and higher order sparse Gauss-Hermite points are used within this algorithm construct. This simplified ES that uses deterministically chosen points is able to achieve excellent performance on challenging multimodal problems, and is competitive with the state-of-the-art CMA-ES. It must also be noted that a parallel island implementation of this new USA-ES is developed and applied to the lunar lander problem

in this chapter as another unique contribution. All of these results indicate that this contribution may likely prove to be a very useful tool in addressing the challenges associated with multimodality and guess sensitivity that continue to cause difficulty in the arena of optimization and optimal control.

Lastly, Chapter 7 takes this idea several steps further by introducing unscented sampling techniques into the state-of-the-art NES. This allows for the full utilization of the unscented abscissas where both the nodes and the associated weights are used to evolve both the distribution mean, and the distribution covariance. This is a major, unique contribution that has shown great promise in solving the challenging set of benchmark problems from literature. Multimodality and guess sensitivity pose difficult challenges that plague many optimization algorithms. However, empirical results indicate that this new uxNES is able to find globally optimal solutions on the many of multimodal problems. Furthermore, it is able to consistently find globally optimal solutions to many of these challenging problems with uniform random initial guesses, indicating insensitivity to the initial guess for this set of problems. The results from this chapter illustrate a new algorithm that is potentially able to address both major issues of multimodality and guess sensitivity on many notoriously difficult problems from literature. This new uxNES algorithm is then integrated into a global optimization scheme using the concepts of a reigning optimal and fitness scaling to further enhance the capabilities of the uxNES. This is a unique contribution in and of itself that produces promising empirical results warranting further research and development. Lastly, this new uxNES written in Matlab code, produces near-optimal results for a lunar lander problem much faster than other evolutionary methods investigated without the use of parallel processing. These results illustrate that this is another promising new algorithm that may prove to be very useful for challenging optimization and optimal control problems.

8.3 Relevant Astrodynamical Applications

There are several astrodynamical engineering applications that the new evolutionary algorithms in this dissertation are particularly well-suited for. Some of the most challenging problems to solve, especially when using gradient-based optimization techniques, are problems with both discrete and continuous decision variables, referred to as hybrid optimization problems. There are a fairly large number of optimization problems that fall into this category of hybrid optimization within the realm of astronautical engineering. As previously

stated in Chapter 1, tasks such as optimizing satellite constellation designs [12], [13], finding optimal deep-space mission designs with multiple flybys for gravity assist benefits [11], [23], and satellite or launch vehicle design optimization [8], [9] are all real-world applications that lead to the presence of both discrete and continuous decision variables that need to be optimized.

An increasingly important hybrid optimization problem is the determination of optimal (or near-optimal) data collection schedules for a single satellite, multiple satellites in a constellation, or even across multiple constellations of different satellite systems that are oversubscribed, meaning they have more desired collects than possible [19]. This quickly becomes a very complex, dynamic, hybrid optimization problem, even when considering the most simplified case of an oversubscribed single satellite. Assuming the satellite of interest has already been designed and has a fixed set of capabilities, there are still a very large number of considerations that must be accounted for in order to optimize this particular satellite's collection routine. For the sake of exercise, it can be assumed that the satellite's primary mission is a commercial Earth imaging satellite. There are a number of constraints that must be considered when optimizing a collection schedule for this oversubscribed system. There are revisit limitations, where the satellite is on a set orbit with a set field of view creating limited windows where a given target on the Earth is observable by the satellite sensor. Next, there are collection time requirements, where the sensor on the satellite requires sufficient time to sweep over a target (e.g. Landsat requires 24 seconds per collect) [21]. On-board data storage coupled with down-link data bandwidth and ground station locations can create additional limits regarding collection thresholds. In addition, slew rate capability can determine the transition time from one target to the next. Pointing angles need to be accounted for, as too large an angle may result in unacceptable image resolution and quality. Thermal control and power restrictions can also influence how many slews and in what direction the satellite can face for given amounts of time. Cloud cover and weather conditions can dictate whether or not an image can be collected during a given window. These are just a few of the considerations outlined by Globus et al. when considering Earth observing satellite scheduling [21]. It can be imagined that there are a number of complicating factors that arise when moving targets are considered. To further develop this problem, there are many applications where all targets are not created equal. Targets may have associated value functions that can be static, or can change with

time. As an example, a customer may be willing to pay a very high price for a particular image collected at a specific sun angle, however, this same customer may still want this image collection at a reduced price if that desired sun angle is not exactly met. It can be seen that only accounting for a few of the larger factors of an imaging satellite scheduling problem using a single satellite can quickly result in a very challenging, hybrid optimization problem. A number of papers have pointed toward stochastic-based algorithms to attempt to optimize these types of collection problems based on their ability to work in discontinuous and discrete search spaces [19]–[22].

The visual range of the electromagnetic spectrum is a very small piece of a very large pie. There are a number of other sensors and satellite missions out there that all have their own unique constraints and requirements. The afore-mentioned earth observing collection schedule optimization problem, while very complex and difficult in its own right, may be perceived as a small piece of a much larger problem. In order to get a much more detailed analysis and understanding of a given target, there may be a desire to use multiple sensors to determine various aspects of a given target or given set of targets. There may be sets of collection requests for specific sensors, or there may be collection requests that span across multiple sensors that could potentially involve multiple satellite systems in different constellations. This optimization of using multiple sensors to generate a more complete story of target sets with time-varying value functions associated with each target is a very difficult problem to optimize. The number of constraints continues to grow with varying sets of unique constraints associated with each sensor type, satellite orbit, target requirement, etc. This results in a very highly constrained, high-dimensional, non-differentiable, dynamic, hybrid search space that will prove to be challenging for any optimization technique. However, the evolutionary algorithms introduced and developed in this dissertation have proven to be effective in solving fairly high dimensional problems with non-differentiable search spaces. With this, the satellite collection optimization problem may be a very interesting and rewarding application for these new unscented evolutionary algorithms. Of course, as with any complex problem, it is suggested that a very simple version of the problem is initially used and slowly built upon to be increasingly representative of real world systems.

8.4 Further Development of the Unscented RCGA

The unscented RCGA introduced and summarized in Appendix C is primarily used in this work as a stepping stone toward the unscented ES variants. A number of ideas and further development can be done regarding unscented sampling techniques within the construct of RCGAs. This section outlines a few ideas that should be investigated, but is likely just scraping the surface of a much larger group of innovative ideas and techniques.

8.4.1 Unscented RCGA Selection Operators

The version of the unscented RCGA utilizes the DGX crossover operator where 3rd order sigma points are used for sampling rather than drawing two offspring points at random from a Gaussian distribution. However, this results in $2n + 1$ potential offspring from each set of parent vectors leading to the requirement of narrowing down this set of sample points to only two offspring. Initially, a truncation selection technique is used, where the two best performing offspring are chosen. This requires that all of the potential offspring be evaluated and sorted for each set of parent vectors that undergoes crossover. Unfortunately, this technique requires a large amount of computational effort and may lead to less diversity within the overall GA population. One way around this would be to utilize a selection operator that introduces an element of randomness, such as tournament selection. Perhaps a pure random selection from the unscented sample points would be an effective way of removing the requirement to evaluate the fitness for all potential offspring, and introduce more diversity into the population. A study that further develops the selection operator type utilized within the unscented DGX crossover operator would advance this idea and could further enhance its performance.

8.4.2 Recombination vs Selection

Closely linked to the idea of studying various selection operators within the unscented DGX crossover operator, various recombination techniques that are typically used in ESs could be implemented to reduce the number of offspring for each set of parents down to two. Intermediate recombination, or fitness-based recombination techniques could be utilized, making the individual DGX crossover operator similar to a USA-ES. The unscented RCGA would essentially become a population of modified USA-ESs. However, the two offspring that result from this new form of unscented crossover operator are then mixed back into the

population and undergo all of the standard GA operators such as tournament selection and mutation. It would be interesting to see how this form of the unscented RCGA compares with the parallel USA-ES developed in Chapter 6.

8.5 Further Development of the USA-ES

While the initial idea of the USA-ES and closely related GHSA-ES have produced very promising results throughout this dissertation, there are still a number of ideas and modifications that should be further investigated and developed. These algorithms are still very much in their infancy and have a lot of room in terms of growth and performance enhancement. This section points out several recommendations and ideas that could lead to further improvement of these algorithms.

8.5.1 Different USA-ES Cooling Schedules

A simple, exponentially decaying covariance cooling schedule is used for the USA-ES and GHSA-ES in this dissertation. However, numerous other cooling schedules could be used to control how the covariance evolves over time. Anything from a linear decay to a oscillating function such as a sinusoidal curve could be explored. It is likely that the performance of various cooling schedules and associated cooling rates is dependent on the characteristics of the problem being solved. With this, it may make sense to run these algorithms on a given problem multiple times with different cooling schedules to analyze what type of cooling schedules and rates work the best for certain problem characteristics.

8.5.2 USA-ES Search Distribution Types

All of the variations of the USA-ES utilize Gaussian distributions throughout this dissertation, as these distributions are the most commonly used search distribution for ESs. They are often used due to their compliance with Beyer's mutation operator guidelines, along with their ability to closely emulate biological mutation processes [151]. However, a number of other distribution types could also be parameterized using deterministically chosen points in a similar fashion as is done with the 3rd order sigma points and sparse Gauss-Hermite points for Gaussian distributions. As an example uniform distributions and heavy-tailed Cauchy distributions may allow better exploration characteristics for certain fitness landscapes. In addition, beta distributions have additional parameters that may give

the user/algorithm more flexibility to change how the algorithm searches. The possibilities are endless, and these are just a few ideas of different search distributions that could be parameterized and implemented within the context of a USA-ES or GHSA-ES. As stated before, and in accordance with the No Free Lunch Theorems, it is likely that different search distributions will perform very favorably on certain classes of problems and very unfavorably on others.

8.5.3 Parallel USA-ES With Multiple Distribution Types

A new parallel USA-ES (or GHSA-ES) could be developed in a way that allows for each processor to run a version of the algorithm that utilizes a different distribution type. This could significantly increase the algorithm robustness, as the performance of certain search distribution types are likely problem dependent. Along these same lines of thinking, different covariance cooling schedules could also be implemented on each processor. This would be a fairly simple algorithm to implement using a parallel island architecture, as has been done throughout this dissertation. This could leverage the benefits of parallel processing in a way that could greatly enhance the robustness of this algorithm so that its performance would be less dependent on problem type and fitness landscape characteristics.

8.5.4 USA-ES/GHSA-ES Theory

While a great deal of empirical evidence, along with some intuitive analysis of how the USA-ES/GHSA-ES perform favorably when compared to Monte Carlo based methods is presented in this dissertation, the mathematical theory behind why the application of deterministic sampling in the USA-ES enhances the performance still needs to be fully developed. The paper by Auger et al. may offer a starting point to this analysis, as they are able to theoretically prove that their mirrored sampling techniques can obtain faster convergence rates on spherical cost functions [201]. Their mirrored sampling idea within the construct of an ES is conceptually the closest documented ES sampling technique to the deterministic sampling techniques used within the USA-ES and GHSA-ES that has been uncovered in the literature survey. Unfortunately, there are some notable differences, that will need to be addressed and modeled mathematically in order to further develop this theoretical performance gain that has been shown empirically.

8.6 Further Development of The uxNES

As with the USA-ES, there are several modifications and adaptations that can be done to potentially improve the robustness and performance of the uxNES on various types of problems. This section outlines top-level suggestions and concepts regarding several of these ideas. It would be naive to claim that these algorithms are perfect, as it is known that they perform better on certain classes of problems than they do on others. With this, there are always going to be improvements that could lead to more robust and effective uxNES algorithms for various problem types.

8.6.1 uxNES Distribution Types

The uxNES developed and investigated in this dissertation utilizes Gaussian search distributions that are parameterized with deterministically chosen sample points. Gaussian distributions are used given their popularity and ability to emulate biological mutation processes, however, these algorithms can work with other types of distributions as well. Various classes of problems may be more aptly solved with different search distributions. There are different unscented sampling techniques that can be used for the various forms of distributions. In fact, there is no guarantee that the Gaussian distributions produce the best results on the multimodal benchmark problems and lunar lander optimal control problem investigated in this dissertation. This also relates back to the No Free Lunch Theorem discussed in Section 1.4 where it has been statistically proven that no single algorithm can produce superior results across all classes of problems. Lastly, a parallel version of the uxNES could be developed in a way that allows for different processors to not only start in different locations of the search domain, but could each use different search distribution types.

8.6.2 New Stopping Criteria Within uxNESs

One of the issues that can be challenging for evolutionary algorithms is finding an effective stopping criteria that can indicate an optimal solution has been found [206]. Gradient-based solvers are able to look at whether or not KKT conditions are satisfied, and some innovative ideas have been proposed for using similar approaches that approximate KKT conditions in evolutionary algorithms [87], [88], [206]. This approach can add unwanted complexity to the evolution strategies that can somewhat mitigate the initial simplicity

of these algorithms. However, the variations of the NES algorithms are using search gradient information to update parameters. Perhaps it would be feasible to develop a new set of stopping criteria that leverages information from the search gradients already being calculated within the algorithms. A pseudo set of KKT conditions is one idea, but perhaps there are others that could be developed and lead to a more meaningful stopping criteria for these types of evolutionary algorithms.

8.6.3 uxNES With Self-Tuning Learning Rates

As was investigated in Section 7.7, the learning rates for the uxNES can play a fairly large role in algorithm performance. Self-tuning learning rates for covariance and mean adjustments have been investigated and proposed for the standard NES [41]. However, this method would need some modification to effectively self-tune the learning parameters for the variations of the NES that utilize deterministically chosen sampling points. Successful development and implementation of self-tuning learning rates within the construct of an uxNES could significantly increase algorithm robustness and remove the need for the tedious process of tuning learning rates through empirical experimentation.

8.7 Development and Application of The Unscented CMA-ES

It has been shown in several papers that the CMA-ES is directly related to the NES. [41], [49], [191] Furthermore, it seems to be accepted by a few authors that the CMA-ES can outperform the NES on several problems [49], [191]. With this, it seems that the CMA-ES could be modified to utilize the deterministic sampling and quadrature techniques that are applied throughout this dissertation to potentially further improve this state-of-the-art ES. While the integration of unscented sampling points may be a little less straightforward than it is for the NES variations, it is likely feasible and could enhance the performance of the CMA-ES on some classes of problems. This may require some re-work of the CMA-ES algorithm from the bottom up where some of the constants and parameters may need to be re-derived and adjusted with the use of deterministic sampling in mind. This could lead to a new form of unscented CMA-ES that may advance the current state of one of the leading ESs.

8.8 Implementation of Unscented Sampling In Other Stochastic Algorithms

While GAs and ESs may be the most straightforward application of unscented sampling within evolutionary algorithms, there are potentially other forms of evolutionary algorithms that could also benefit from the integration of these ideas. Relatively new algorithms that have been introduced in the 1990s and have recently been grouped into the realm of evolutionary algorithms such as particle swarm, differential evolution, and ant colony optimization are prime candidates. This section briefly investigates the possibilities of incorporating unscented sampling techniques within the construct of each of these algorithm types.

8.8.1 Particle Swarm Optimization

Algorithm 8 illustrates a standard global particle swarm optimization scheme that models a simplified social system [207]. It can be seen that this algorithm operates on populations of solution vectors in a similar way to GAs and ESs. However, this algorithm separately tracks the local reigning optimal solution for each “particle,” or solution vector, as it evolves over time. In addition, the algorithm also tracks the reigning optimal for the global population. It then uses a velocity term derived from a combination of the particles local performance, coupled with the algorithms global performance to update each current particle’s position within the search space. In the below algorithm c_1 and c_2 are user-defined constants. In addition U-Rand() indicates a uniform random number between zero and one and g is the

current generation number.

Algorithm 8: Particle Swarm Optimization Algorithm

Initialize a population of particles (solution vectors) with uniform random positions, \bar{x}_i and velocities, \bar{v}_i ;

while stopping criteria not satisfied **do**

Evaluate fitness for each particle;

Track each particle's best fitness over time (position and value) and update if current value if necessary ($pbest$);

Track the overall populations best global fitness (position and value) and update current value if necessary ($gbest$);

Update particle velocity, \bar{v}_i , and position, \bar{x}_i , according to the following (c_1 and c_2 are user-defined constants): ;

$$\bar{v}_i^{(g+1)} = \bar{v}_i^{(g)} + c_1 \times \text{U-Rand}() \times (\bar{x}_{pbest,i} - \bar{x}_i^{(g)}) + c_2 \times \text{U-Rand}() \times (\bar{x}_{gbest,i} - \bar{x}_i^{(g)});$$

$$\bar{x}_i^{(g+1)} = \bar{x}_i^{(g)} + \bar{v}_i^{(g+1)};$$

$$g = g + 1;$$

end

While it may not be as straightforward as it was with ESs, this algorithm can likely benefit from the introduction of deterministic sampling. One idea would be to replace the uniform random variables used to update the velocity for each particle with a set of deterministically chosen points and weights. The velocity term can then be updated using a weighted sum of the deterministically chosen uniform points. This of course is just one idea, where there are likely to be numerous modifications that could allow for the introduction of deterministic sampling. A more obvious application would be to use deterministically chosen points and weights for the initialization step of the algorithm where the initial position and velocity of each particle is assigned. In addition, a combination of these two ideas can be implemented for a new form of particle swarm optimization algorithm.

8.8.2 Differential Evolution Optimization

Differential evolution is a newer class of stochastic optimization algorithm that was introduced by Storn and Price in 1997, and is summarized in Algorithm 9 [208]. This algorithm steps through each member of the population during each generation setting it as the target

vector referred to \bar{x}_{target} . It then randomly selects three unique vectors from the population and uses them to create a “donor” vector, \bar{V} , in a step that these authors refer to as mutation. Next, uniform crossover is performed between the target vector and this donor vector to generate a trial vector, \bar{U} . The fitness values of both the new trial vector and the target vector are compared. If the new trial vector performs better than the target vector, it replaces the target vector in the population. The algorithm then moves on to the next vector in the population and sets it as the new target vector. After each member of the population has been set as a target vector, the generation number is incremented and the algorithm repeats this process. The only user-defined parameters in this algorithm are a crossover probability

p_{cross} and a weight factor W .

Algorithm 9: Differential Evolution Algorithm

Initialize a population of particles (solution vectors) with uniform random positions, \bar{x}_i ;

Define weight parameter W and crossover probability p_{cross} ;

while stopping criteria not satisfied **do**

for target=1 to population size **do**

Mutation Step:;

Randomly select 3 vectors from the population;

Create a donor vector \bar{V} ;

$$\bar{V} = \bar{x}_{r1}^{(g)} + W \left(\bar{x}_{r2}^{(g)} - \bar{x}_{r3}^{(g)} \right);$$

;

Crossover Step:;

Create a trial vector \bar{U} ;

for $i=1$ to n **do**

δ = Uniform Random[0, 1];

if $\delta \leq p_{cross}$ **then**

$\bar{U}_i = \bar{V}_i$;

end

else

$\bar{U}_i = \bar{x}_{target,i}^{(g)}$;

end

end

Evaluate fitness of \bar{U} and $\bar{x}_{target}^{(g)}$;

if $F(\bar{U}) \leq F(\bar{x}_{target}^{(g)})$ **then**

$\bar{x}_{target}^{(g+1)} = \bar{U}_i$;

end

else

$\bar{x}_{target}^{(g+1)} = \bar{x}_{target}^{(g)}$;

end

end

$g = g + 1$;

end

This algorithm will likely require some modification for a practical implementation of an unscented sampling technique to be incorporated into this construct. However, one fairly simple idea would be to use the target vector as the mean of a Gaussian distribution from which sigma points could be determined. From this step, either the top three sigma points are selected, or three randomly selected sigma points are used for the remainder of the differential evolution process. This unscented technique would replace the standard method of randomly selecting three vectors from the current population in a way that could potentially add another level of sophistication to the algorithm. In addition, this could better explore the search space by introducing new sample points, rather than only working with vectors that exist within the current population. This would require the user to either define a fixed covariance for the Gaussian distributions, use a covariance cooling schedule, or develop a more advanced covariance evolution scheme. This new algorithm would essentially be a fusion between some of the unscented ESs developed in this dissertation with the differential evolution algorithm. This would likely increase the computation time required for each generation, however, it may still increase the overall performance of the algorithm in terms of accuracy and efficiency.

8.8.3 Ant Colony Optimization

Ant colony optimization algorithms are very interesting in that they emulate the social behavior of certain types of ants that transpires as colonies forage for food. They were initially introduced in the early 1990s and have since been applied to a number of combinatorial type problems [209]. These algorithms utilize “pheromones” to let other “ants” know where the most promising search paths lie. Traditionally, these algorithms have been designed for discrete variables and combinatorial problems where the solutions can be pieced together from a finite number of options. Unfortunately, this makes them difficult to work with in terms of integrating unscented sampling techniques within these algorithms. However, the application of ant colony optimization algorithms to continuous problems is an ongoing area of research. [209] With this, it is recommended that a close eye is kept on these techniques as they develop. This could also prove be a promising area of research with a potential application of unscented sampling.

8.9 Conclusion

This dissertation has overturned a new leaf in terms of exploring the potential applications of unscented sampling techniques within the arena of evolutionary algorithms. It has introduced and explored several new algorithms that are able to produce very promising results when solving challenging NLPs. These algorithms are all well-suited for the incorporation of parallel computing to further enhance both computational speed and accuracy. While a simple lunar lander optimal control problem is used to demonstrate the effectiveness of these algorithms on an engineering application, the hope is that these algorithms will prove to be of great benefit for a number of real-world astrodynamical applications. The developments and techniques presented in this dissertation are poised to be a springboard for a number of new and innovative evolutionary algorithms to come. This research does not make the claim that any of the new evolutionary algorithms introduced in this work are in their optimal form, and admits that there is likely room for improvement. While this dissertation has introduced several original concepts and ideas pertaining to evolutionary algorithms, the hope is that it will lead to an entirely new path for evolutionary algorithm research and development that will greatly benefit the astrodynamical engineering community among others.

APPENDIX A: Definition of Symbols By Chapter

A.1 Chapter 1 Symbol Definitions

Table A.1. Chapter 1 symbol definitions

\bar{x}	Set of state variables
\bar{u}	Set of control variables
t	Time
t_0	Initial time (usually zero)
t_f	Final time
\bar{x}_f	Final state
n	Problem dimension
$\dot{\bar{x}}$	Time derivative of state variables
$J[]$	Cost function/cost functional
$E()$	Mayer cost
$\int F()$	Lagrange cost
$e()$	Endpoint manifold
N_x	Number of state vector variables
N_u	Number of control vector variables
N_t	Number of time nodes
$g_i()$	Inequality constraints
$h_i()$	Equality constraints
l	Number of inequality constraints
m	Number of equality constraints
UB	Vector of upper bounds
LB	Vector of lower bounds
X_n	Given state of an algorithm

A.2 Chapter 2 Symbol Definitions

Table A.2. Chapter 2 symbol definitions

$g()$	Gravitational model
r	Radial position from earth's center
μ	Geocentric longitude
λ	Geocentric latitude
V	Velocity magnitude
γ	Flight path angle
ξ	Heading (clockwise from east)
\dot{r}	Time derivative of r
$\dot{\mu}$	Time derivative of μ
$\dot{\lambda}$	Time derivative of λ
\dot{V}	Time derivative of V
$\dot{\gamma}$	Time derivative of γ
$\dot{\xi}$	Time derivative of ξ
α	Vehicle angle of attack
σ	Bank angle
m	RLV empty mass
$J[]$	Cost function
$E()$	Mayer cost
$\int F()$	Lagrange cost
\bar{x}	State vector
\bar{u}	Control vector
$\dot{\bar{x}}$	Time derivative of \bar{x}
$e()$	Endpoint manifold
$h()$	Path constraints
S_{ref}	RLV reference area
σ_0	Standard density
r_{ref}	Reference altitude
β	Inverse scale height
GM	Earth's gravitational constant
Ω	Earth's angular velocity
R_e	Earth's Radius
h_0	Initial RLV Altitude

$\rho()$	Atmospheric density model
ρ_0	Atmospheric density at sea level
$C_D()$	Coefficient of drag model
$C_L()$	Coefficient of lift model
D	Drag force
L	Lift force
M	Mach number
$a()$	Speed of sound
h	Altitude above sea level
η_z	Lateral g force
\bar{q}	Dynamic pressure
Q	Heat flux
w	Weight parameter
DU	Distance scaling unit
MU	Mass scaling unit
VU	Velocity scaling unit
TU	Time scaling unit
n	Problem dimension or size
p	Number of processors
T^*	Shortest time for serial execution of a code
S_p	Speedup of code going from serial to parallel
S_{max}	Maximum speedup
f	Fraction of code that must be executed in serial
T_p	Time required to run code in parallel

A.3 Chapter 3 Symbol Definitions

Table A.3. Chapter 3 symbol definitions

n	Problem dimension
\bar{x}	Solution vector or “chromosome”
$f()$	Objective or “cost” function
ϵ	Small positive value
$bits$	Number of bits in a string

$value_{max}$	Maximum value represented by <i>bits</i>
$value$	Unscaled value represented by a bit string
$bound_{upper}$	Upper bound on a variable
$bound_{lower}$	Lower bound on a variable
$variable_{scaled}$	Scaled variable within $[bound_{lower}, bound_{upper}]$
$resolution$	Resolution of a variable represented by a binary string
\oplus	Exclusive OR operator
\otimes	Exclusive AND operator
x_{bi}	Binary encoded variable
x_{gr}	Reflective Binary Gray encoded variable
p_{cross}	Crossover probability
p_{mut}	Mutation probability
*	Wildcard bit (could be either 0 or 1 for binary)
l	Chromosome length (number of bits)
O_s	Schemata order (number of defining bits)
l_s	Schemata defining length between leftmost and rightmost defining bits
N_{pop}	Number of chromosomes in a population (population size)
N_s	Number of a given schema in a population
g	Current generation number
f_s	Average fitness of a given schema in a population
\bar{f}	Average population fitness
p_{diff}	Probability that two selected parents are different
$p_i(y)$	Probability that a given population will produce chromosome y
$m_{i,j}(0)$	Probability of producing all zeros from a given set of parents i and j
$M_{i,j}$	Matrix whose i th and j th components are equal to $m_{i,j}(0)$
σ_j	Permutation column vector
r	Maximum number of unique chromosomes of a given length
F	Diagonal matrix where $F_{i,i}$ is equal to $f(\bar{x}_i)$
ϕ_i	Incidence vector of length 2^l for binary where each element is the number of times \bar{x}_i occurs in a population
N_{tot}	Maximum number of unique combinations of chromosomes for a given population size (order does not matter)
$Q_{i,j}$	Probability transition matrix for a simple GA (N_{tot} by N_{tot} matrix)

Z	The state space for a simple GA (N_{tot} by r matrix)
π	Probability of each possible chromosome being present in the population after a specified number of iterations
P_0	Initial population
f^*	Globally optimal fitness value
$\bar{x}_{best}(t)$	Best solution found in an elitist GA at time t
$f(\bar{x}_{best}(t))$	Best fitness value found in an elitist GA at time t
D_t	Difference between best fitness found at time t and globally optimal fitness
$E[g(\delta)]$	Expected number of generations required to have a δ probability of finding the globally optimal solution
K	Encoding cardinality
f_{max}	Best fitness found so far for a maximization problem
f'	Better fitness value between two selected parent chromosomes
k_1, k_2, k_3, k_4	User-defined learning rates for self tuning p_{cross} and p_{mut} functions
k_{bb}	Building block length
Q_{bb}	Number of correctly chosen building blocks
σ_{bb}	Standard deviation of fitness values of chromosomes containing a given building block
d_{bb}	Difference in mean fitness between the two competing building blocks
$P^{(1)}$	Parent chromosome number one
$P^{(2)}$	Parent chromosome number two
$c_i^{(1)}$	The i th component of parent chromosome 1
$c_i^{(2)}$	The i th component of parent chromosome 2
$O^{(1)}$	Offspring chromosome number one
$O^{(2)}$	Offspring chromosome number two
α	Uniform random number between 0 and 1
$c_{max,i}$	Maximum i th parameter from a set of parents
$c_{min,i}$	Minimum i th parameter from a set of parents
I_i	Difference between $c_{max,i}$ and $c_{min,i}$
β	Uniform random number between 0 and 1
λ	Uniform random number between 0 and 1
ω	Uniform random number between 0 and 1
k_p	Desired number of parents

y	Desired number of offspring
G	Geometric center of parents
$Y^{(j)}$	The j th simplex point
ζ	User-defined constant
ξ	Random normally distributed parameter
η_j	Random normally distributed parameter
G_m	Midpoint between two parents
d_p	Difference vector between two parents
D_{p3}	Distance between $P^{(3)}$ and the line connecting $P^{(1)}$ and $P^{(2)}$
η	User-defined spread parameter for SBX
u	Uniform random number between 0 and 1
γ	Probability from an SBX distribution
τ	Uniform random variable between 0 and 1
g_{max}	Maximum number of generations
LB_i	Lower bound on the i th component
UB_i	Upper bound on the i th component
α_i	Lower bound of a sub-interval between LB_i and UB_i
β_i	Upper bound of a sub-interval between LB_i and UB_i
s	RCGA schema
P	Number of processors
T_p	Total computation time required
T_f	Computation time required for a single function evaluation

A.4 Chapter 4 Symbol Definitions

Table A.4. Chapter 4 symbol definitions

\bar{x}	State vector
$u(t)$	Thrust trajectory as a function of time t
$h(t)$	Altitude as a function of time t
$v(t)$	Vertical velocity as a function of time t
$m(t)$	Spacecraft mass as a function of time t
c_g	Gravitational acceleration constant
c_v	Specific impulse constant

Δt	Time step size
N_t	Number of discretized time nodes
t_f	Final time
$\dot{\bar{x}}$	Time derivative of state vector
$\bar{x}_{k+1}^{(s1)}$	RK3 numerical time integrator stage 1 at time node $k + 1$
\bar{h}	Discretized altitude vector N_t nodes
\bar{v}	Discretized vertical velocity vector N_t nodes
\bar{m}	Discretized mass vector N_t nodes
\bar{u}	Discretized thrust vector N_t nodes
u_0	Initial thrust value
h_0	Initial altitude
m_0	Initial mass
v_0	Initial vertical velocity
h_f	Final altitude
v_f	Final vertical velocity
$f()$	NLP objective function
$g_i()$	Inequality constraints
h_k	Equality constraints
l	Number of inequality constraints
m	Number of equality constraints
\mathbb{F}	Feasible set
n	Problem dimension
p	Penalty coefficient
$P()$	Measure of constraint violation
$F()$	Objective function with penalty function included
q	Type of error norm (e.g. L2 error norm is $q = 2$)
F_{L1}	Objective function with L1 type penalty function
F_{L2}	Objective function with L2 type penalty function
F_{Linf}	Objective function with infinity norm penalty function
F_{Quad}	Objective function with quadratic type penalty function
$\#_{constraints}$	Total number of constraints in an NLP
$\#_{infeasible}$	Total number of violated constraints in a given set of GA trials
$infeas_{avg}$	Average feasibility measurement across a set of GA trials

\bar{x}_{trial}	Best solution found from a GA run for a given trial
p_{mut}	Mutation probability
p_{cross}	Crossover probability
Mig./MaxGen	Number of migrations per total number of generations
g	Generation number
tol	User defined tolerance for equality constraints
β_1	User-defined dynamic penalty decay rate
β_2	User-defined dynamic penalty growth rate
$P^{(1)}$	Parent chromosome number one
$P^{(2)}$	Parent chromosome number two
$c_i^{(1)}$	The i th component of parent chromosome 1
$c_i^{(2)}$	The i th component of parent chromosome 2
$O^{(1)}$	Offspring chromosome number one
$O^{(2)}$	Offspring chromosome number two
LB_i	Lower bound on the i th component
UB_i	Upper bound on the i th component
λ	Uniform random number between 0 and 1
ω	Uniform random number between 0 and 1
σ_0	Initial standard deviation in each dimension
$\sigma(g)$	Standard deviation in each dimension at a given generation
η	User-defined spread parameter for SBX
α	“Cooling” rate parameter for σ or η
u	Uniform random number between 0 and 1
γ	Probability from an SBX distribution

A.5 Chapter 5 Symbol Definitions

Table A.5. Chapter 5 symbol definitions

\bar{x}	Solution vector
\bar{x}^*	Optimal solution vector
$f()$	Objective function
$f(\bar{x})$	Objective function value
$f(\bar{x}^*)$	Optimal objective function value

μ	Number of parent vectors in an ES
λ	Number of offspring vectors in an ES
$+$;	Notation signifying either parents compete (+) or parents die out at each generation (,)
\bar{x}_{parent}	Parent solution vector and mean of parent-centric distribution
σ	Single standard deviation parameter for all dimensions (spherical)
C_{parent}	Covariance matrix for a given parent
ρ	Subset size of selected offspring to undergo recombination
g	Generation number
n	Problem dimension
$\bar{x}_{offspring}^{(k)}$	The kth offspring vector
$\bar{z}^{(k)}$	The kth random vector selected from a parent-centric distribution
$f_{N(0,1)}(z_i)$	1-D Gaussian probability density function
σ_i	Indicates different standard deviations in each dimension
I	Identity matrix
A	Cholesky factor of the C matrix
G	Matrix whose columns are orthonormal Eigenvectors of C
D^2	Matrix whose diagonal values are the Eigenvalues of C
$\bar{x}_{\frac{\mu}{\rho}}$	Parent solution vector formed from recombination
$\bar{x}^{(i)}$	The ith selected vector used for recombination
$F(\bar{x}^{(i)})$	Fitness of the ith selected vector used for recombination
$w_{i:\rho}$	Weight associated with the ith of ρ selected and sorted vectors in fitness rank order
$\tilde{w}_{i:\rho}$	Normalized $w_{i:\rho}$
$\bar{x}^{(i:\rho)}$	The ith of ρ selected and sorted vectors in fitness rank order
$f_{corridor}$	Corridor fitness function
f_{sphere}	Spherical fitness function
\circ	Component-wise product operator
k	Offspring number
τ_0	Global learning parameter for Self-Adaptive covariance technique
τ	Local learning parameter for Self-Adaptive covariance technique
d	User-defined constant for adaptive covariance techniques
d_1	User-defined constant for adaptive covariance techniques

\bar{s}_σ^{g+1}	Evolution path vector for covariance with no off-diagonal terms at generation $g + 1$
c_σ	Constant for updating the evolution path vector
\bar{s}_C^{g+1}	Evolution path vector for full covariance at generation $g + 1$ excluding \bar{s}_σ^{g+1}
w_k	Normalized logarithmically shaped fitness recombination weights
μ_ρ	Fitness weight normalizing factor
c_C	C Full covariance evolution path coefficient
c_1	C update coefficient
c_ρ	C update coefficient
c_h	C update coefficient
c_m	\bar{x}_{parent} mutation coefficient
θ	Vector of Gaussian distribution parameters (\bar{x}_{parent}, C)
$\pi(\bar{x}_{offspring} \theta)$	Gaussian probability density function
$E[]$	Expectation
$J(\theta)$	Fitness expectation
$\nabla_\theta J(\theta)$	Gradient of fitness expectation w.r.t. θ (search gradient)
\bar{x}_k	The kth n -dimensional random sample taken from $\mathcal{N}(\bar{x}_{parent}, C)$
$\nabla_{\bar{x}_{parent}} \log \pi(\bar{x}_{offspring} \theta)$	Gradient of the log-density function w.r.t. \bar{x}_{parent}
$\nabla_C \log \pi(\bar{x}_{offspring} \theta)$	Gradient of the log-density function w.r.t. C
$\tilde{\nabla}_\theta J(\theta)$	Natural gradient of expected fitness w.r.t. θ
F	Fischer information matrix
$\hat{w}_{k:\lambda}$	Weirstra et al. version of normalized logarithmically shaped fitness-based weights
δ	Distribution mean mapped to a natural exponential frame
M	Covariance mapped to a natural exponential coordinate frame
B	A factorized A matrix such that $A = B\sigma$
$\nabla_\delta J(0, 0)$	Gradient of expected fitness at $(\delta = 0, M = 0)$ w.r.t. δ
$\nabla_M J(0, 0)$	Gradient of expected fitness at $(\delta = 0, M = 0)$ w.r.t. M
$\nabla_B J(0, 0)$	Gradient of expected fitness at $(\delta = 0, M = 0)$ w.r.t. B
$\nabla_\sigma J(0, 0)$	Gradient of expected fitness at $(\delta = 0, M = 0)$ w.r.t. σ
$\eta_{\bar{x}_{parent}}$	Learning rate for adjusting the parent vector or mean
η_C	Learning rate for adjusting the covariance matrix

η_σ	Learning rate for covariance step size
η_B	Learning rate for covariance matrix
ϵ	Small vicinity near a global optimum
ϕ	Local progress rate
\bar{Q}	Quality gain

A.6 Chapter 6 Symbol Definitions

Table A.6. Chapter 6 symbol definitions

μ	Number of parent vectors in an ES
λ	Number of offspring vectors in an ES
ρ	Subset size of selected offspring to undergo recombination
n	Problem dimension
g	Generation number
C	Covariance matrix
A	Cholesky factor of the C matrix
\bar{x}	Solution vector
κ	Unscented sigma point user-defined spread factor
χ_i	ith n-dimensional unscented sigma point
$\bar{\chi}_i$	ith n-dimensional unscented sigma point that has been shifted by a mean and scaled by a covariance
$f()$	Objective function
\bar{x}^*	Optimal solution vector
$\bar{x}_{\frac{\mu}{\rho}}$	Parent solution vector formed from recombination
$\bar{x}^{(i)}$	The ith selected vector used for recombination
$F(\bar{x}^{(i)})$	Fitness of the ith selected vector used for recombination
$S()$	Check function to prevent division by zero for fitness weighted recombination
eps	Matlab machine precision value
$\bar{x}_{parent}^{(g)}$	Parent solution vector at generation (g)
g_{max}	Maximum number of generations
$w_{i:\rho}$	Weight associated with the ith of ρ selected and sorted vectors in fitness rank order
$\tilde{w}_{i:\rho}$	Normalized $w_{i:\rho}$

$\bar{z}^{(k)}$	kth random vector selected from an n-dimensional standard normal distribution
$\bar{x}_{offspring}^{(k)}$	kth offspring solution vector
Z	Z-score for desired confidence interval
CI	Confidence interval
σ	Standard deviation

A.7 Chapter 7 Symbol Definitions

Table A.7. Chapter 7 symbol definitions

n	Problem dimension
κ	Unscented sigma point user-defined spread factor
χ_i	ith n-dimensional unscented sigma point
C	Covariance matrix
σ	Standard deviation
$\bar{\chi}_i$	ith n-dimensional unscented sigma point that has been shifted by a mean and scaled by a covariance
$W_{\sigma,i}$	Weight associated with the ith unscented sigma point
μ	Parent vector and mean of a search distribution
θ	Vector of Gaussian distribution parameters (μ, C)
$E[]$	Expectation
$J(\theta)$	Fitness expectation
$\nabla_{\theta}J(\theta)$	Gradient of fitness expectation w.r.t. θ (search gradient)
λ	Number of n -dimensional sample points used
$f()$	Objective or fitness function
$\pi(\bar{x} \theta)$	Probability density function of a Gaussian distribution
$\nabla_{\theta}\log\pi(\bar{x} \theta)$	Gradient of the Gaussian log density function w.r.t. θ
$\nabla_{\mu}\log\pi(\bar{\chi}_k \theta)$	Gradient of the Gaussian log density function w.r.t. the mean μ
$\nabla_C\log\pi(\bar{\chi}_k \theta)$	Gradient of the Gaussian log density function w.r.t. the covariance C
LB	Lower bound
UB	Upper bound
$F_{\bar{x}_{parent}^{(g)}}$	Fischer information matrix w.r.t. the mean \bar{x}_{parent} at generation g
$F_{C^{(g)}}$	Fischer information matrix w.r.t. the covariance C at generation g
η_{μ}	Learning rate for adjusting the parent vector or mean

η_C	Learning rate for adjusting the covariance matrix
A	Cholesky factor of C
δ	Distribution mean mapped to a natural exponential coordinate frame
M	Covariance mapped to a natural exponential coordinate frame
σ_x	Covariance step size for uxNES
B	A factorized A matrix such that $A = B\sigma_x$
$\nabla_\delta J(0, 0)$	Gradient of expected fitness at $(\delta = 0, M = 0)$ w.r.t. δ
$\nabla_M J(0, 0)$	Gradient of expected fitness at $(\delta = 0, M = 0)$ w.r.t. M
$\nabla_B J(0, 0)$	Gradient of expected fitness at $(\delta = 0, M = 0)$ w.r.t. B
$\nabla_{\sigma_x} J(0, 0)$	Gradient of expected fitness at $(\delta = 0, M = 0)$ w.r.t. σ_x
η_σ	Learning rate for covariance step size
η_B	Learning rate for covariance matrix
$\hat{f}()$	Scaled fitness
$ F _{max}$	Maximum absolute value of population fitness values
b	Scale factor
ϵ	Required fitness improvement in the global optimization algorithm
$\bar{x}_{reigning}$	Reigning optimal solution vector
$f_{reigning}$	Reigning optimal fitness value
p	Penalty parameter
α	Rate of change of ϵ
$fail_{max}$	Maximum algorithm iterations where $f_{reigning}$ is not improved by ϵ

A.8 Chapter 8 Symbol Definitions

Table A.8. Chapter 8 symbol definitions

\bar{x}_i	Solution vector
\bar{v}_i	Velocity vector associated with the i th solution vector
p_{best}	Best solution and fitness for a given particle over time
g_{best}	Best solution and fitness for a population over time
c_1	User-defined constant
c_2	User-defined constant
g	Current generation number
U-Rand()	Uniform random number between 0 and 1

\bar{x}_{target}	Current solution vector in the population that is being updated
\bar{V}	A donor vector created from a combination of 3 randomly selected vectors from the population
\bar{U}	Candidate vector formed from \bar{V} and \bar{x}_{target} to potentially replace \bar{x}_{target}
W	User-defined weight parameter
p_{cross}	Crossover probability
n	Problem dimension
$F()$	Fitness function

A.9 Appendix B Symbol Definitions

Table A.9. Appendix B symbol definitions

\bar{x}	Solution vector
$f()$	NLP objective function
$g_i()$	Inequality constraints
h_k	Equality constraints
l	Number of inequality constraints
m	Number of equality constraints
LB	Lower bound
UB	Upper bound
n	Problem dimension
p	Penalty coefficient
$P()$	Measure of constraint violation
$F()$	Objective function with penalty function included
q	Type of error norm (e.g. L2 error norm is $q = 2$)
$s_i(\bar{x})$	Scaled constraint violation
R_i	Penalty parameter that changes with user-defined levels of constraint violation
g	Current generation number
α	User-defined exponent
β	User-defined integer term
C_1	User-defined constant
$SVC()$	Sum of violated constraints

$D_i()$	Measure of equality constraint violations
$D_k()$	Measure of inequality constraint violations
$Q()$	Dynamic penalty function
$R(\alpha, \beta)$	Simulated annealing penalty function
c_i	Growth rate parameter
P_i	Dynamic penalty term
σ	Standard deviation of population fitness values
f_{max}	Maximum fitness in a population
f_{min}	Minimum fitness in a population
$\tilde{f}(\bar{x})$	Normalized fitness value
$c_j(\bar{x})$	Constraint violation measure
$c_{max,j}$	Maximum constraint violation for a solution vector in the population
$v(\bar{x})$	Normalized constraint violation measure
r_f	Ratio between feasible solutions and the total population for each generation
T	Number of generations before updating penalty coefficient
p_{min}	Minimum allowed penalty coefficient
p_{max}	Maximum allowed penalty coefficient
φ_g	Current ratio of feasible solutions to population size
φ_{min}	User-defined minimum ratio of feasible solutions to population size
C_{BFS}	Cost of best feasible solution
C_{BIS}	Cost of best infeasible solution
∇v_i	Constraint violation measure
NFT_i	Near feasibility threshold value
λ	User-defined decay rate
V_{all}	Best fitness value found
V_{feas}	Best feasible fitness value found
k	User-defined parameter
v_j	Violation measure for a single constraint
k_j	Normalized penalty factor
N_{pop}	Number of vectors in a population
R	Penalty parameter
σ_i	Multiplier parameter
τ_i	Multiplier parameter

λ_i	Lagrange multiplier
μ_k	Lagrange multiplier
$\nabla_{\bar{x}}$	Gradient matrix
V	Constraint violation vector
ΔV	Change in constraint violation vector
pop_1	Population 1
p_1	Penalty parameter for amount of constraint violation
pop_2	Population 2
p_2	Penalty parameter for number of constraints violated
F_{pop_2}	Average fitness of feasible solutions found in pop_2
F_{pop_1}	Equals F_{pop_2} obtained from a given penalty parameter setting
v_{dist}	Amount of constraint violation
v_{count}	Number of constraints violated

A.10 Appendix C symbol definitions

Table A.10. Appendix C symbol definitions

n	Problem dimension
κ	Sigma point spread parameter
p_{cross}	Crossover probability
p_{mut}	Mutation probability
F_{Best}	Best fitness found
N_{pop}	Population size

APPENDIX B: Survey of Constraint Handling Techniques For Evolutionary Algorithms

While standard evolutionary algorithms can be very effective in solving difficult optimization problems, they are only applicable to unconstrained problems. The standard operators that are used in most evolutionary algorithms can often result in infeasible solutions when solving constrained optimization problems. This results in algorithms that don't lend themselves to constrained optimization without some additional techniques to handle constrained problems. A perfectly unconstrained optimization problem is seldom (if ever) found when working with real-world applications due to the existence of physical limitations. Given the usefulness of evolutionary algorithms and the constrained nature of most optimization problems, there has been a significant amount of research and progress made with a number of constraint handling techniques. The primary challenge associated with all of these constraint handling techniques is that their effectiveness is often times dependent on the problem being solved in addition to the algorithm type used. This is still a very active research area, given that a robust, problem-independent evolutionary algorithm constraint handling technique has yet to be truly developed and applied. This appendix provides the reader with an overview of a number of different techniques that can be used within evolutionary algorithms for constraint handling. This is not all inclusive, but does try to highlight a number of the more popular techniques. For further reading, there are a number of good survey papers that introduce a number of various constraint handling techniques for both GAs and ESs [177], [210], [211]. For the purposes of this Appendix, nonlinear programming problems that can be written in the standard notation depicted in equation B.1 are used to illustrate the various constraint handling techniques.

$$\begin{aligned} & \text{minimize } f(\bar{x}) \\ & \text{subject to:} \\ & g_i(\bar{x}) \leq 0 \quad i = 1, \dots, l \\ & h_k(\bar{x}) = 0 \quad k = 1, \dots, m \end{aligned} \tag{B.1}$$

Where l is the number of inequality constraints, m is the number of equality constraints, and $\bar{x} \in \mathbb{R}^n$ is a n-dimensional solution vector. In addition to this, the plausible search space \mathbb{R}^n is often defined by bounds on the state variables $\bar{x}^{UB} > \bar{x} > \bar{x}^{LB}$. The feasible set \mathbb{F} is a subspace of \mathbb{R}^n that is created by the intersection of all constraints and state bounds $(g_i(\bar{x}) \cup h_k(\bar{x}) \quad \forall i, k \text{ s.t. } \bar{x} \in \mathbb{R}^n)$.

B.1 Penalty Functions

Penalty functions are the oldest and most widely used methods for constraint-handling in evolutionary algorithms [210]. They transform a constrained optimization problem into an unconstrained optimization problem through the construction of a fitness function $F(\bar{x})$ that is typically a variation of the general form outlined in Equation B.2. It is often created from a linear combination of the objective function $f(\bar{x})$ and a penalized measure of constraint violation. The measure of constraint violation can be something as simple as the number of constraints violated, or some determination of the distance from an infeasible solution to the feasible region. It is interesting to note that most of the research and development effort has been placed on finding appropriate values of p [212]–[215] and not on experimenting with different violation measurement techniques.

$$\text{minimize } F(\bar{x}) = f(\bar{x}) + p * P(\bar{x}) \quad (\text{B.2})$$

There are lots of variations to this general form, however, it is important to first distinguish some of the known classifications of penalty functions. There is a concept of exactness in penalty functions that is investigated by Di Pillo and Grippo. They provide a formal definition of exactness for a large group of penalty functions described by Equation B.3 below [172]. These penalty functions are based on norms of both the inequality and equality constraints in the form shown in Equation B.1. The authors are able to create a specific definition of exact penalty functions and prove exactness of the penalty functions in Equation B.3 for both convex and non-convex spaces. Exact penalty functions have several characteristics that make them very useful to these types of applications. At a high level, an exact penalty function is one where the variables of the unconstrained problem formulation lie in the same space as the variables of the original constrained problem. In addition, the optimal solution to an exact penalty function is the same as the optimal solution to the

original constrained problem.

For $1 \leq q < \infty$

$$F_q(\bar{x}) = f(\bar{x}) + p * \left[\sum_{i=1}^l (\max[0, g_i(\bar{x})])^q + \sum_{k=1}^m |h_k(\bar{x})|^q \right]^{\frac{1}{q}} \quad (\text{B.3})$$

For $q = \infty$

$$F_\infty(\bar{x}) = f(\bar{x}) + p * \max[\max[0, g_1(\bar{x})], \dots, \max[0, g_l(\bar{x})], |h_1(\bar{x})|, \dots, |h_m(\bar{x})|]$$

Di Pillo and Grippo further distinguish between differentiable and non-differentiable penalty functions. Due to the fact that penalty functions have their roots in gradient-based search techniques, a great deal of effort has been spent on using differentiable penalty functions. Equation (B.8) below illustrates a quadratic form that doesn't fit within the group of exact penalty functions defined in Equation (B.3) given that it is not in the form of a standard Euclidean norm. However, it has been used in a number of papers [173]. One of the main luxuries of GAs is that they do not require differentiable or continuous problems. Given this, a lot of effort can be saved in simply using non-differentiable forms of exact penalty functions.

$$F_q(\bar{x}) = f(\bar{x}) + p * \left[\sum_{i=1}^l (\max[0, g_i(\bar{x})])^2 + \sum_{k=1}^m |h_k(\bar{x})|^2 \right] \quad (\text{B.4})$$

Richardson et al. investigated several different static penalty function formulation methods to develop a set of general guidelines to be used in penalty function construction [216]. After looking at penalties based on the number of violated constraints versus more sophisticated penalties that accounted for the amount of constraint violation and repair costs, Richardson et al. were able to come to the following guidelines and conclusions:

1. Penalties that are functions of Euclidean distance from the feasible set perform better than those that are simply functions of the number of violated constraints.
2. For problems having few constraints and few feasible solutions, penalties that are functions of the number of constraint violations will not likely find solutions.
3. Good penalties can be constructed from two quantities: the expected completion cost (a mean fitness cost associated with making the infeasible solution feasible based on a distribution of possible repairs) and the maximum completion cost (the upper bound

on most expensive scenario of repairing infeasible solution).

4. Penalties should be close to the expected completion cost without frequently falling below it. The more accurate the completion cost calculation, the better the penalty. A penalty that underestimates the completion cost will have difficulty finding solutions.

Where Richardson et al. determine the expected completion cost as a distance from the feasible region based on an average problem slope. The maximum completion cost is a conservative estimate of the distance to the feasible region based on the maximum problem slope. While these metrics are based on problems where derivative information is available, they are still useful in illustrating the above conditions. Another way to view these results is that the penalty factor has to be appropriately tuned so that the best solution found isn't an infeasible one. However, the penalty factors can't be too large, or they will fail to search the boundaries of the feasible region where the true optimal solutions typically lie. Richardson also recommended the exploration of dynamic penalty factors that start small and grow throughout the progression of the genetic algorithm.

B.1.1 Static Penalty Functions

Static penalty functions are setup so that everything is defined and set before the evolutionary algorithm is executed. In other words, the penalty parameters don't change after the code is running.

Seywald and Kumar use the most basic of static penalty functions, however, they investigate the differences between two constraint violation measurement techniques [164]. Equation (B.5) below illustrates that they look at both the absolute value of the constraint violation (equality constraint) and the maximum function of the constraint violation (inequality constraint).

$$\begin{aligned} F(\bar{x}) &= f(\bar{x}) + p * \sum_{k=1}^m |h_k(\bar{x})| \\ F(\bar{x}) &= f(\bar{x}) + p * \sum_{i=1}^l \max[0, g_i(\bar{x})] \end{aligned} \tag{B.5}$$

After running through three fairly simple optimization problems with both methods, Seywald and Kumar conclude that efficiency is significantly increased when the constraints are

converted from equality constraints to inequality constraints. However, these authors do not investigate the effect different values of p have on the afore mentioned outcome. Most papers on penalty functions indicate that the penalty factor must be carefully tuned for each problem to achieve efficient results with static penalty functions.

Homaifar et al. developed a systematic approach to vary the penalty parameter R_i with respect to user-defined levels of constraint violation [212]. Similar to the above methods, Homaifar converts a constrained nonlinear programming problem into an unconstrained one through the use of penalty functions as can be seen in Equation (B.6).

$$\begin{aligned} \text{minimize } F(\bar{x}) &= f(\bar{x}) + P(\bar{x}) \\ P(\bar{x}) &= \begin{cases} 0 & x \in \mathbb{F} \\ \sum_{i=1}^l s_i(\bar{x})R_i & x \notin \mathbb{F} \end{cases} \end{aligned} \quad (\text{B.6})$$

where $s_i(\bar{x})$ is a scaled constraint violation term that has been scaled according to a user-defined reference value and R_i is a penalty parameter that is dependent on $s_i(\bar{x})$. The user-defined reference value that is used to scale the constraint violations is illustrated 2 separate ways: 1.) using local reference values for each constraint or 2.) using a single, global reference value for all constraints. Given the reference value, the user also has to define various levels of constraint violation and corresponding R_i values associated with each level of constraint violation. This method is demonstrated on three nonlinear programming problems using both local and global reference values. The results obtained from a GA using this method are compared to a general gradient search method along with the known reference solutions to each problem. While Homaifar's method does preform well for the given problems, it is quickly observed that its effectiveness is dependent on the problem being solved. More specifically, a lot of tuning and adjustment is required to go from one problem to the next. The user has to redefine a global reference value (or multiple, local reference values), establish an appropriate set of violation ranges, and correctly tune the corresponding penalty parameter values that are to be associated with each level of violation for each constraint. It does not take too much imagination to see how this process can quickly become cumbersome as the number and complexity of constraints grows. In addition, the extensive use of switching statements used to determine the various levels of constraint violation can result in longer run times and less computational efficiency. Regardless of the previously mentioned issues, Homaifar et al. helped lay the foundation for

a number of more recent adaptive penalty techniques and provided one of the first examples of a scaled, adaptive penalty technique designed specifically for GAs.

B.1.2 Dynamic Penalty Functions

Dynamic penalty functions are a specific group of penalty methods that change in one way or another as a function of the number of generations or progression of the evolutionary algorithm. It has been observed by several authors that it is often good practice to allow more infeasible solutions to occur earlier on in the evolution process, and then slowly force more and more feasible population members as the code progresses [217]–[220]. Dynamic penalty methods use various techniques to achieve this result.

Joines and Houck were some of the first to introduce penalty functions that change during a single run of a evolutionary algorithm [213]. Their method uses a fairly common penalty function that is dependent on the amount of constraint violation. In addition, they increase the penalty parameter as a function of the generation number. In Equation (B.7), $\rho_g = C_1 g$ with g being the number of generations, α is a user-defined exponent, and C_1 is a user-defined constant. In addition, $D_i()$, is a measure of equality constraint violations, $D_k()$, is a measure of inequality constraint violations, and $SVC()$ is the sum of violated constraints. Lastly, $Q()$ is the resulting dynamic penalty associated with this method. While this is a linear penalty parameter growth rate (when $\alpha = 1$), Joines and Houck also investigate a quadratic growth rate where $\alpha = 2$. In addition, an exponential penalty parameter growth rate that is more in-line with a simulated annealing approach $R(\alpha, \beta) = e^{P(\alpha, \beta)}$ is investigated. It must be noted that this form of penalty function doesn't fit into the forms of exact penalty functions defined by Di Pillo and Grippo given that it does not use standard norms [172].

$$\begin{aligned}
D_i(\bar{x}) &= \begin{cases} 0 & g_i(\bar{x}) \leq \epsilon \\ |g_i(\bar{x})| & \text{otherwise} \end{cases} \\
D_k(\bar{x}) &= \begin{cases} 0 & -\epsilon \leq h_k(\bar{x}) \leq \epsilon \\ |h_k(\bar{x})| & \text{otherwise} \end{cases} \\
SVC(\beta, \bar{x}) &= \sum_{i=1}^l D_i^\beta(\bar{x}) + \sum_{k=1}^m D_k^\beta(\bar{x}), \quad \beta = 1, 2, \dots \\
Q(\alpha, \beta) &= \rho_g^\alpha SVC(\beta, \bar{x})
\end{aligned} \tag{B.7}$$

Kazarlis and Petridis further investigated the effects and performance associated with different penalty parameter growth rate profiles [214]. They utilize a very similar method to Joines and Houck, however, they expand on this idea by looking at seven different penalty parameter growth rate functions. Kazarlis and Petridis use a common GA and integrate the various growth rate functions to solve a cutting stock problem and a unit commitment problem. After comparing the results, the authors are able to note that the functions with a slow initial growth rate seem to consistently outperform penalty functions that have a rapid, initial growth rate. These findings are in agreement with a number of studies that have determined infeasible chromosomes can carry useful information and often result in better performance when compared to methods that quickly eliminate all infeasible chromosomes in a population [217]–[220]. This improved performance is often attributed to the idea that most optimal solutions lie on a boundary between feasible and infeasible search spaces, or in other words, at least one of the constraints are active at the optimal point. Given this observation, it is often beneficial to search near, or on these constraint boundaries. For evolutionary algorithms, it becomes difficult to produce offspring that are on this boundary, if all members of the population lie within the feasible region. Another way to think about it is in the light of multiple, disjoint feasible regions. If the search space is quickly reduced to one of these feasible regions, a local optimal will be reached without much chance of chromosomes migrating to other feasible regions that may contain the global optimal solution. In addition to the benefit of a slow initial growth rate, Kazarlis and Petridis were able to conclude that the optimal growth rate is problem-dependent. What works best for the cutting stock problem, isn't the function that works best for the unit commitment problem. In addition, penalty growth rates that work well on one problem, perform poorly on others. Again, the challenge of problem dependency and lack of a uniform constraint-handling technique that performs well for a large set of problems has yet to be overcome.

Crossley and Williams also investigate several penalty parameter growth rates, however, they investigate the effects of three different constraint violation functions [221]. Equation (B.8) is the quadratic constraint violation function commonly used due to its roots from calculus-based methods. Realizing that evolutionary algorithms don't require continuously differentiable functions due to their heuristic methods they also investigate a linear penalty and a step penalty outlined in Equation (B.9) and Equation (B.10) respectively. In this method, c_i is the growth rate parameter. During this study, Crossley and Williams investigate

the following growth rates: 2 , g , g^2 , 2^g , σ , and σ^2 . In this notation, g , is the generation number, σ is the standard deviation of population fitness values, and σ^2 is the variance of population fitness values. It must be noted that the last two growth rates could be viewed as adaptive penalty methods, as they are utilizing information about the population to tune the penalty parameters. More specifically, they increase the penalty parameter when the fitness of the population varies greatly from member to member. Crossley and Williams use a standard total fitness function defined by $F(\bar{x}) = f(\bar{x}) + \sum_{j=1}^{l+m} P_j(\bar{x})$. After comparing each of these penalty techniques over three test problems, they essentially find that various combinations of these methods perform better for various problem formulations. It was noted that the standard, linear coefficient $c_i = 2$, combined with the quadratic penalty function in Equation (B.8) performed the best, when looking at the normalized average fitness values across all three problems.

$$P_i = c_i [\max(0, g_i(\bar{x}))]^2 \quad (\text{B.8})$$

$$P_i = c_i [\max(0, g_i(\bar{x}))] \quad (\text{B.9})$$

$$P_i = \begin{cases} 0 & \text{if } g_i(\bar{x}) \leq 0 \\ c_i[1 + g_i(\bar{x})] & \text{else} \end{cases} \quad (\text{B.10})$$

B.1.3 Adaptive Penalty Functions

Adaptive penalty methods are slightly different than dynamic penalty methods in that they use some type of feedback based on various metrics to adjust the penalty parameters. Typically, some measurement of the percentage of the population that is feasible, or some relative measurement is used for these types of penalty techniques. It must be noted that there have been several techniques that couple this adaptive feedback technique with a generation-dependent dynamic penalty technique as well [220], [222].

Tessema and Yen have developed an adaptive penalty function that has an interesting algorithm to adjust the penalty parameters based on the number of infeasible solutions in the population [215]. These authors utilize an effective method of normalizing both the objective function $\tilde{f}(\bar{x})$ and the amount of constraint violation $v(\bar{x})$ so that all values of the population vary between 0 and 1 as illustrated in Equation (B.11). In this formulation,

$c_j(\bar{x})$ is a constraint violation measurement, $c_{max,j}$ is the maximum constraint violation for a solution vector, and r_f is the ratio between feasible solutions and the total population for each generation. It can be seen that $v(\bar{x})$ is essentially the L1 norm penalty function form that has been normalized and averaged.

$$\begin{aligned}\tilde{f}(\bar{x}) &= \frac{f(\bar{x}) - f_{min}}{f_{max} - f_{min}} \\ v(\bar{x}) &= \frac{1}{l+p} \sum_{j=1}^{l+m} \frac{c_j(\bar{x})}{c_{max,j}}\end{aligned}\tag{B.11}$$

Where:

$$c_j(\bar{x}) = \begin{cases} \max(0, g_j(\bar{x})) & j = 1, \dots, l \\ \max(0, |h_j(\bar{x})| - \epsilon) & j = l+1, \dots, l+m \end{cases}$$

$$c_{max,j} = \max c_j(\bar{x})$$

Equation (B.12) depicts how the fitness function $F(\bar{x})$ is calculated using all of this information for three different cases. It can be seen that this method only evaluates and uses the scaled objective function, $\tilde{f}(\bar{x})$, if there are members of the population that are feasible ($r_f \neq 0$).

$$F(\bar{x}) = \begin{cases} v(\bar{x}) & r_f = 0 \\ \tilde{f}(\bar{x}) & \text{For feasible members} \\ \sqrt{\tilde{f}(\bar{x})^2 + v(\bar{x})^2} + [(1 - r_f)v(\bar{x}) + r_f\tilde{f}(\bar{x})] & \text{otherwise} \end{cases}\tag{B.12}$$

The unique addition, other than the normalization of objective function and constraint violations, is the fact that the objective function isn't considered in the fitness function if there are no feasible solutions in the population. Given this, in a fully infeasible population, the infeasible solutions that have the lowest constraint violation will be selected to generate the next population with the idea that they will eventually evolve to a feasible region. In addition, an infeasible chromosome that is in a partially feasible population will be penalized based on how much of the population is feasible. As an example, if the population is almost all feasible, then the infeasible solution won't be penalized as heavily as it would be if the majority of the population was infeasible. This is done to create a good balance of infeasible and feasible solutions in the population. It is also interesting to note that there are no user-

defined parameters required for the fitness function described in Equation (B.12) which is an advantage to this method. Tessema and Yen use twenty-two benchmark problems to illustrate that this method is competitive with a several other constraint handling techniques and requires less user-defined parameters.

Wu and Simpson implement a boundary searching, self-adaptive penalty method that they test on a water distribution network optimization problem [219]. Their method is a little interesting in that it is a hybrid between a co-evolutionary method and an adaptive penalty method in that they use ‘feedback’ from population metrics to adapt the penalty factor, however, they also represent the penalty factor as a decision variable in the evolutionary algorithm population. Throughout the evolution process, they track the amount of feasible and infeasible solutions in the population. Every T generations, they use the ratio of feasible solutions to the total population to adjust the range $[p_{min}, p_{max}]$ that they set for the penalty factor decision variable within the evolutionary algorithm according to Equation (B.13). They are essentially using Di Pillo and Grippo’s L1 norm method to determine the amount of constraint violation for each chromosome.

$$\begin{aligned}
 & \text{if } g_t - g_{t-1} \geq T \text{ and } \varphi_g < \varphi_{min} \\
 & \quad p_{min,t} = (1.0 + \alpha)p_{min,t-1} \\
 & \quad p_{max,t} = (1.0 + \alpha)p_{max,t-1} \\
 & \text{if } g_t - g_{t-1} \geq T \text{ and } \varphi_g \geq \varphi_{min} \\
 & \quad p_{min,t} = (1.0 - \alpha)p_{min,t-1} \\
 & \quad p_{max,t} = (1.0 - \alpha)p_{max,t-1}
 \end{aligned} \tag{B.13}$$

Where φ_g is the current portion of feasible solutions in the population and φ_{min} is the user-defined minimum desired ratio. In addition, α is a user-defined value between 0 and 1 that dictates how fast the limits or box constraints on the penalty parameter are adjusted. The penalty factor is only adjusted every T generations with the notation of g_t and g_{t-1} indicating the current adjustment generation and the previous generation that the penalty parameter range $[p_{min}, p_{max}]$ was adjusted. In Wu and Simpson’s method, the user has to: initially define the range $[p_{min}, p_{max}]$, a desired minimum ratio of feasible solutions φ_{min} , define the frequency of adjusting the parameters T , and define how fast they want the limits

on the penalty factor to be adjusted α . Given this, Wu and Simpson do illustrate that this self-adaptive co-evolution hybrid method yields superior results on a water distribution network optimization problem when compared to a GA with fixed penalty factors.

Afshar and Mariño have developed a ‘parameter-free’ self-adapting penalty function that they use to optimize a pipe network problem [220]. Similar to Wu and Simpson, this penalty parameter attempts to drive the GA in a direction that searches the boundaries between feasible and infeasible search spaces. Describing that most constrained optimization problems have at least one active constraint at the optimal solution, this seems to be a logical approach. They illustrate that the ratio between the fitness or penalized cost of the best feasible solution C_{BFS} and the fitness of the best infeasible solution C_{BIS} can be a useful metric in determining the appropriate penalty parameter size. Equation (B.14) below illustrates how they use this to adapt their penalty parameter as the evolutionary algorithm evolves the populations.

$$p^{g+1} = p^g \left(\frac{C_{BFS}}{C_{BIS}} \right) \quad (\text{B.14})$$

In addition to this, they also illustrate that the user can choose an exterior method by choosing a very low initial value of p^1 . In other words, the GA will initially spend most of its time searching the infeasible region, and then work its way toward the feasible region. Using the same logic, the user can start with an interior penalty method by choosing a very large value of p^1 so that the GA initially spends more search effort in the feasible region and works its way toward the infeasible region. In either case, the penalty parameter will migrate toward a value that allows for both C_{BFS} and C_{BIS} to be close to equal. This is another way of ensuring that there are both infeasible and feasible members of the population, or searching along the constraint boundaries.

Coit and Smith define a near-feasibility threshold (NFT) parameter that is used to help steer the search along areas that are close to the constraint boundaries in their adaptive penalty method [222]. Referring to Equation (B.15), it can be seen that each amount of constraint violation or ∇v_i is scaled by the NFT_i parameter. This parameter decreases throughout the evolutionary algorithm progression as a function of g (generation number), an initial user-defined NFT value $NFT_{o,i}$, and a user-defined decay rate λ . The term that makes this an adaptive penalty instead of a standard dynamic penalty is $V_{all} - V_{feas}$, which is the difference between the best overall objective function value found and the best feasible

objective function value found. It is also noted that this value can't be negative, in other words, if no constraints are active, this value is set to zero.

$$\begin{aligned}
 F(\bar{x}) &= f(\bar{x}) - \sum_{i=1}^l \left(\frac{\nabla v_i}{NFT_i} \right)^k (V_{all} - V_{feas}) \\
 NFT_i &= \frac{NFT_{o,i}}{1 + \lambda g} \\
 \nabla v_i &= |g_i(\bar{x})|
 \end{aligned} \tag{B.15}$$

The authors originally set this up for a reliability allocation problem and they test this method on 33 variations of these types of problems. Coit and Smith show that it outperforms a penalty method that simply removes any infeasible solutions from the population, referred to as a death penalty method. In addition the authors compare several fixed values of the NFT with the dynamic version outlined above. Coit and Smith's results illustrate that the performance of this penalty method is dependent on values of NFT and is likely dependent on values of other tunable, user-defined parameters to include: $NFT_{o,i}$, k , and λ .

Barbosa and Lemonge describe another adaptive penalty method that is free of user-defined, tunable parameters that automatically adjusts the penalty parameter for each individual constraint instead of grouping them all together as many techniques do [223]. Their method distinguishes between constraints in a way that penalizes difficult constraints more than the ones that seem to be easily satisfied based on population feasibility.

Equation (B.16) illustrates their proposed adaptive penalty method. It must be noted that v_j is the violation for a single constraint and not a sum across all constraints, given that each constraint is handled separately. Next, it can be seen that the penalty factor k_j is essentially the violation of a given constraint j across the current population that is normalized the total violation of all the constraints across the population, this value is then multiplied by the sum of all fitness values $f(\bar{x})$ of the population. Where there are l inequality constraints and m equality constraints, and N_{pop} is the number of individuals in the population. Barbosa and Lemonge use a standard, gray coded GA to compare this method to several other penalty function methods on a number of the benchmark test problems that have become somewhat standard when comparing penalty techniques. Their results illustrate that this method is comparable to the adaptive penalty methods that they compare it to on the given set of

problems.

$$\begin{aligned}
F(\bar{x}) &= \begin{cases} f(\bar{x}) & \bar{x} \in \mathbb{F} \\ f(\bar{x}) + P(\bar{x}) & \bar{x} \notin \mathbb{F} \end{cases} \\
P(\bar{x}) &= \sum_{j=1}^{l+m} k_j v_j(\bar{x}) \\
v_j(\bar{x}) &= \begin{cases} \max[0, g_j(\bar{x})] & \text{inequality} \\ |h_j(\bar{x})| & \text{equality} \end{cases} \\
k_j &= \frac{\left| \sum_{s=1}^{N_{pop}} f(\bar{x}_s) \right|}{\sum_{h=1}^{l+m} \left[\sum_{s=1}^{N_{pop}} v_h(\bar{x}_s) \right]^2} \sum_{s=1}^{N_{pop}} v_j(\bar{x}_s)
\end{aligned} \tag{B.16}$$

Deb and Srivastava introduce a GA augmented Lagrangian method that also has a self-adapting penalty value [224]. This method is similar to a penalty function approach, however; Deb and Srivastava are able to extract the optimal Lagrange multipliers, λ_i and μ_k , from the result with the additional cost of multiple optimization problems being solved at each step. They also indicate that the augmented Lagrangian method distorts the original problem less than standard penalty methods. When compared to these standard penalty techniques, the augmented Lagrangian method is less likely to inadvertently create multimodality and nonlinearities that were not existent in the original problem. For their specific implementation, Deb and Srivastava use a standard NLP solver (MATLAB's fmincon) to solve the sub-level optimization problem. Equation (B.17) depicts the generic form of a quadratic augmented Lagrangian function that they utilize in their algorithm.

$$\begin{aligned}
F(\bar{x}, \sigma^t, \tau^t) &= f(\bar{x}) + R \sum_{i=1}^l [(\max\{0, g_i(\bar{x}) + \sigma_i^t\})^2 - (\sigma_i^t)^2] + R \sum_{k=1}^m [(h_k(\bar{x}) + \tau_k^t)^2 - (\tau_k^t)^2] \\
\sigma_i^{t+1} &= \max\{\sigma_i^t + g_i(\bar{x}^t), 0\}, \quad i = 1, 2, \dots, l \\
\tau_k^{t+1} &= \tau_k^t + h_k(\bar{x}^t), \quad k = 1, 2, \dots, m
\end{aligned} \tag{B.17}$$

Their algorithm adjusts the penalty parameter R every so many iterations, t , as a function of the total population constraint violation and the average population fitness. In addition to this, it also scales the values of the multiplier parameters σ and τ with the adjusted R value by the ratio of $\frac{R_{new}}{R_{old}}$. After each generation, their algorithm utilizes a NLP solver to further optimize the best chromosome found in that generation. This is done to help ensure convergence to a true KKT point. Finally, once an optimal solution is found, the optimal Lagrange multipliers can be found using Equation (B.18).

$$\begin{aligned}\lambda_i &= -2R\sigma_i^T \\ \mu_k &= -2R\tau_k^T\end{aligned}\tag{B.18}$$

B.2 Genetic Operator Techniques

Deb uses an enhanced tournament selection method to enforce the three criteria listed below using a tournament size of two individuals [225].

1. Between an infeasible solution and a feasible one, the feasible solution is selected.
2. Between two feasible solutions, the one with the best fitness value is selected.
3. Between two infeasible solutions, the one with the least amount of constraint violation is selected.

Furthermore he also uses a slightly different fitness function that penalizes infeasible solutions without the use of a penalty factor. Equation (B.19) describes the relationship where f_{worst} is the worst fitness of the feasible solutions. It can be seen that the amount of constraint violation is added to f_{worst} to ensure that all infeasible solutions will have a fitness that is worse than that of a feasible solution. It is also important to note that Deb normalizes the constraints to prevent any one constraint being favored over others during the GA run. He implements this technique on a binary-coded GA and on a real-coded GA. In addition, the results are compared with a penalty function technique by Powell and Skolnick in the first few problems [226]. Deb compares several versions of his technique on nine optimization problems to conclude that the proposed technique is more efficient and robust than the Powell and Skolnick method in addition to the best constraint-handling methods proposed in literature. However, it is important to note that penalty parameters

must be tuned for each specific problem with some of the other methods, and there isn't an indication that the optimal penalty factors were selected for the comparisons in the first several benchmark problems.

$$F(\bar{x}) = \begin{cases} f(\bar{x}) & \text{if } \bar{x} \in \mathbb{F} \\ f_{worst} + \sum_{i=1}^l |g_i(\bar{x})| & \text{if } \bar{x} \notin \mathbb{F} \end{cases} \quad (\text{B.19})$$

Hinterding and Michalewicz take a similar approach to Deb, however they take it one step further in that they try to match infeasible parents based on which constraints they satisfy [227]. It must be noted that parent matching was originally done by Ronald back in 1995, however his method didn't look at collective constraint satisfaction among infeasible parents [228]. Their selection operator follows the same guidelines listed above, with the addition of trying to maximize the number of satisfied constraints that the offspring of two infeasible solutions may have. As an example, if there are four constraints and two infeasible parents, the goal would be that both parents collectively satisfy the largest number of constraints possible. So if the first parent satisfied constraints one and two, the selection operator would look for the infeasible parent that satisfied constraints two and three. A simple binary array is used to track which constraints are met by each chromosome. The idea is that this type of parent matching will lead to a better chance of the infeasible solutions producing feasible offspring. Similar to Deb, they use the tournament selection process outlined in the list below to select the first parent.

1. Between an infeasible solution and a feasible one, the feasible solution is selected.
2. Between two feasible solutions, the one with the best fitness value is selected.
3. Between two infeasible solutions
 - (a) Select the one with the lowest number of violated constraints
 - (b) If they violate the same number of constraints, select the one that has lowest constraint violation measure.

From this point, the second parent is chosen with similar rules, however, when selecting between two infeasible solutions, the best compliment to the first parent is chosen using the logic described above.

Michalewicz and Janikow take advantage of convex feasible search spaces created by linearly constrained optimization problems to create genetic operators that ensure all chromosomes are within the feasible region [229]. They refer to their GA algorithm as GENOCOP and there have been several additional versions that incorporate additional capabilities and constraint-handling methods. GENOCOP is a real-coded GA that starts by eliminating all of the equality constraints and converting them all into a simplified set of linear inequalities. From this point, GENOCOP determines a feasible initial population that has a certain percentage of the initial chromosomes on constraint boundaries. At this point Michalewicz and Janikow use three different mutation operators along with a crossover operator to evolve the population in a way that ensures all of the chromosomes remain in the feasible region. The mutation operators consist of a standard uniform mutation, a boundary mutation, and a non-uniform mutation. The boundary mutation ensures that the chromosome mutates in a way that at least one of the constraints $g_i(\bar{x})$ will be equal to zero, or active. The non-uniform mutation is adapted in a way that makes it have less and less of an impact on the chromosome values as the number of generations increases. In addition, the crossover operator uses the properties of linear combinations of points inside of convex hulls to ensure feasible offspring.

Runarsson and Yao introduce the idea of stochastic ranking within the selection mechanism as a robust method to ensure an appropriate balance in dominance between the original objective function and the penalty function [230]. They make several observations on what can happen when comparing two chromosomes for selection while using a penalty method (assuming minimization). If one assumes that x_i is selected over x_{i+1} then there are three possible scenarios that can occur.

1. $f(x_i) \leq f(x_{i+1})$ and $p * P(x_i) \geq p * P(x_{i+1})$ where the comparison is dominated by the objective function.
2. $f(x_i) \geq f(x_{i+1})$ and $p * P(x_i) < p * P(x_{i+1})$ where the comparison is dominated by the penalty function.
3. $f(x_i) < f(x_{i+1})$ and $p * P(x_i) < p * P(x_{i+1})$ where the comparison can't be dominated by either function.

It can quickly be inferred that the selection can switch between objective dominated and penalty dominated depending on what value is used for the penalty factor p . Rather than

trying to appropriately adjust p , or develop an adaptive penalty scheme to arrive at a population with a good balance of feasible and infeasible solutions, Runarsson and Yao simply specify a percentage that they want the selection to be objective function dominated, P_f , versus penalty function dominated. With this value specified, the authors have developed a simple method to effectively guide the search in a way that proves to be efficient for a large number of problems. Runarsson and Yao find that $0.5 > P_f > 0.4$ yields the best results after running their algorithm on thirteen benchmark problems and comparing it to dynamic penalty methods. It is further compared with other methods that use these same benchmark problems in Chootinan and Chen's paper where it can be seen as the highest performing GA constraint handling method on these benchmark problems as of 2006 [178].

B.3 Repair Methods

Chootinan and Chen have developed a unique repair method that corrects a portion of infeasible solutions based on gradient information from the constraints [178]. In this technique they obtain gradient information from the constraints using a forward difference approximation (or exact derivative when possible) that can be used to determine the change in constraint violation with respect to the change in the solution vector. Equation (B.20) illustrates that once the gradient matrix $\nabla_{\bar{x}}$ is determined along with the constraint violation vector V , the change in violation vector ΔV and required change in \bar{x} can be determined. An iterative process is then utilized to repair the infeasible chromosomes iteration by iteration using the pseudo inverse of the gradient matrix $\nabla_{\bar{x}}V^\dagger$. As a fail safe, a large penalty factor is applied if the repair method is unable to efficiently repair a chromosome to satisfy all of the constraints after a given number of iterations.

$$\begin{aligned} V &= \begin{bmatrix} \bar{g} \\ \bar{h} \end{bmatrix}_{(l+m) \times 1} \Rightarrow \nabla_{\bar{x}}V = \begin{bmatrix} \nabla_{\bar{x}}\bar{g} \\ \nabla_{\bar{x}}\bar{h} \end{bmatrix}_{(l+m) \times 1} \\ \Delta V &= \nabla_{\bar{x}}V * \Delta \bar{x} \Rightarrow \Delta \bar{x} = \nabla_{\bar{x}}V^{-1} * \Delta V \\ \bar{x}_{t+1} &= \bar{x}_t + \nabla_{\bar{x}}V^\dagger * \Delta V \end{aligned} \tag{B.20}$$

Chootinan and Chen compare this method to several other competitive methods using a set of eleven benchmark problems that are used in several other GA papers. The results illustrate that this method compares very well with the other methods. The only user-defined parameters in this method are the repair rate or percent chance of repairing an infeasible

solution, and the number of iterations before the large penalty factor is utilized.

B.4 Co-evolution and Multiple Objective Optimization

Paredis is the first to introduce the co-evolutionary approach, with an algorithm that uses a GA to solve a constrained, combinatorial chess piece placement problem [231]. This algorithm is based on a predator-prey model in which the fitness of one population impacts the evolution of the other. The first population consists of a set number of constraints that have fitness values that are dependent on how difficult each constraint is to satisfy. In other words, the most restrictive constraints are favored over the less restrictive constraints. It must be noted that the constraint population doesn't evolve and all of the constraints remain unchanged throughout the GA run. However, their associated fitness values do continuously change based on the afore mentioned criteria. The second population consists of potential chess piece placement solutions that are rewarded on the basis of how many constraints they satisfy. Each of these rankings is based on a number of encounters between members of the two populations. For this specific algorithm, only the last 25 encounters are tracked for fitness determination. Selection operators are used to select individuals from each population and they are compared in what is referred to as an encounter. If the chosen constraint from the first population is satisfied by the selected solution from the second population, then the solution is rewarded and the constraint fitness is penalized. This allows more focus to be placed on the most restrictive constraints and at the same time more attention is given to the best solutions that satisfy the most constraints. The second population that contains the potential solutions does evolve based on the fitness determined from 25 encounters with the constraint population.

Coello Coello utilizes co-evolution to adapt penalty factors in a way that gets rid of user-defined tuning parameters and leaves it up to a GA to determine the best penalty factors for solving numeric optimization problems [232]. He illustrates a case with two populations pop_1 and pop_2 where the first population pop_1 is trying to find the optimum set of penalty factors based on an average fitness that is found using a second population pop_2 . Equation (B.21) below illustrates how pop_1 has two penalty factors p_1 to penalize the amount of constraint violation, v_{dist} , and p_2 to penalize the number of constraints violated, v_{count} . However, it must be noted that in order to calculate the fitness for each member of pop_1 a second population pop_2 must be evolved using p_1 and p_2 from that member of pop_1 .

Once evolution is completed on pop_2 a fitness F_{pop_2} is obtained as an average of all feasible solutions found in pop_2 . After this F_{pop_1} can be calculated for each member of pop_1 and the next generation can be formed. In other words, every single generation of pop_1 requires $\text{size}(pop_1) * g_{\max, pop_2}$ generations to occur. Coello Coello illustrates that this technique performs favorably when compared to several other techniques using several test problems. While it doesn't require any user-defined parameters, it is computationally expensive given that it has to run an entire GA on pop_2 for each member of pop_1 for each generation of pop_1 to find each chromosome fitness.

$$\begin{aligned}
 F_{pop_1,m}(p_1, p_2) &= \sum_{n=1}^{\text{size}(pop_2)} \left(\frac{F_{pop_2,n}(p_1, p_2, \bar{x})}{\text{count}_{feas}} \right) + \text{count}_{feas} \quad \forall \bar{x} \in \mathbb{F} \\
 F_{pop_2,n}(p_1, p_2, \bar{x}) &= f_n(\bar{x}) + (v_{dist}p_1 + v_{count}p_2) \\
 v_{dist} &= \sum_{i=1}^l g_i(\bar{x}) \quad \forall g_i(\bar{x}) > 0 \\
 v_{count} &= \text{number of violated constraints}
 \end{aligned} \tag{B.21}$$

B.5 Mapping and Decoder Methods

Koziel and Michalewicz introduce a technique that uses a homomorphic mapping technique to map convex and non-convex feasible regions to an n-dimensional cube(s) $[-1, 1]^n$ thus making it easy to design GA operators that maintain feasibility [233]. This method is robust and fairly problem independent in that there are no user-defined parameters that need to be tuned. However, the mapping can skew the feasible region in a way that varies depending on what feasible reference point is used. They define an iterative method that helps select a reference point that will not skew the area of the feasible region where the optimal solution lies in a way that helps increase the chances of finding the optimal solution. They also extend this method to non-convex and discontinuous feasible regions. This method requires a feasible reference point that is found by random sampling of the search space. Given a feasible reference point, this method was tested on eleven benchmark test problems and performed reasonably well, with the exception of a problem that has several equality constraints.

B.6 Constraint Sequencing

Asafuddoula, Ray, and Sarker utilize a method known as constraint sequencing where they essentially divide the population into $l + m + 1$ subpopulations that each face a different sequence of constraints [234]. As soon as a chromosome in the sub population violates a constraint, the rest of the constraints that it faces are not evaluated and it is assumed that it violates all of the remaining constraints in terms of how the sub populations are ranked. This helps to insure that the boundaries are approached from different directions, as each sub population has a different sequence of constraints that the chromosomes will be compared against. The best members from each of the sub populations are then chosen for evolution and the process continues as expected. This technique performed very favorable when compared to three other state-of-the-art constraint handling techniques.

B.7 Summary

This appendix has introduced the reader to a number of techniques that have been used to successfully handle constraints in various studies and applications. It must be noted that an apples-to-apples comparison of these techniques is fairly difficult for several reasons. First of all, the performance and effectiveness of each method is dependent on the problem being solved and the algorithm configuration being employed. Next, many of these techniques require a number of tunable parameters to be appropriately adjusted, making it difficult to know whether a given constraint-handling method has been optimally tuned for a given problem. Lastly, there has not been a consistent set of benchmark problems or standard experiment configuration used in the analysis of these techniques, making a side-by-side comparison very difficult.

With this in mind, this dissertation leverages a combination of constraint handling tools including: adaptive penalty techniques, exact penalty functions, and simple repair methods to solve constrained optimization problems with evolutionary algorithms. One of the primary reasons for choosing these techniques is that they are very easily transferable from one type of evolutionary algorithm to the next. This makes it easy when developing, modifying, and analyzing a number of different evolutionary algorithms using constrained optimization problems. Next, the adaptive penalty methods and exact penalty functions have been shown to be very effective in practice for a number of problems [175], [215], [223]. Lastly, it will be seen that the simple repair technique is utilized in this dissertation

to enforce simple box constraints on algorithms that do not have a built-in mechanism for enforcing bounds on decision variables. The results of the studies in this dissertation illustrate that this combination of constraint handling techniques can be very effective in solving constrained problems. This survey of constraint handling techniques is included in this dissertation to illustrate that there are a large number of various methods that could potentially be employed on these types of evolutionary algorithms. The most important takeaway from this survey is that there are many tools available to the practitioner when applying various forms of evolutionary algorithms to constrained optimization problems.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C: An Unscented Genetic Algorithm

This appendix introduces a new contribution to GAs where deterministic sampling is applied within a RCGA crossover operator to produce new offspring solution vectors. This idea is what originally directed this research toward the application of deterministic sampling within the construct of the various ESs. However, it is introduced and developed here in the appendices given that deterministic sampling is a more natural fit within the construct of an ES and its distribution-based mutation operators. As the reader will likely appreciate, this makes for a smoother transition and logical explanation to develop the idea of deterministically choosing points to represent distributions within the context of ESs where Gaussian distributions are typically used. Although, the DGX operator introduced in Chapter 4 is a Gaussian based crossover operator that utilizes two randomly chosen sample points from a Gaussian distribution. Similar in some ways to an ES, the characteristics of the Gaussian distribution in this operator are determined by the parent vectors, where the mean is a fitness-weighted average and the covariance is adjusted according to a cooling schedule. It can quickly be seen that this concept is very similar to the USA-ES that is introduced and developed in Chapter 6.

C.1 Unscented Crossover Operator

The idea of an unscented RCGA stems from the idea of utilizing Julier's 3rd order unscented sigma points, outlined in Equation (6.2), as candidate offspring points within the construct of a DGX crossover operator. This is different from the standard DGX operator where only two points, or offspring vectors, are randomly drawn from an n-dimensional normal distribution. When using the 3rd order sigma points, $2n + 1$ offspring are now chosen from the normal distribution in a deterministic way. In order for this to fit within the construct of the RCGAs that are introduced and developed in Chapter 4, another selection operation is needed to determine which two offspring will be chosen from the group of $2n + 1$. Initially, a simple truncation selection operator is used so that the two best performing offspring out of the $2n + 1$ are chosen. Unfortunately, this mechanism requires that the $2n + 1$ offspring solution vectors be evaluated by applying them to the cost function. This process then

has to be conducted for every set of parents that undergoes crossover to produce two new offspring. As can likely be imagined, this significantly adds to the amount of computational time required for the crossover operation. However, the two selected offspring could be chosen at random without evaluating fitness values, or through any of the other selection operators introduced in Chapter 3. Another idea could be to use one of the selection operators that has some randomness to select two subgroups from the $2n + 1$ offspring. Then one of the recombination operators introduced in Chapter 5 could be applied to create two offspring from these two subgroups. For the purposes of introducing this idea, the naive approach of selecting the two offspring that achieve the best fitness is used.

Again, the idea behind using the deterministically selected sample points over randomly chosen points is that a more meaningful sample can be drawn. With the case of 3rd order sigma points, the statistical moments up to 3rd order can be matched with a very small number of sample points. While a very large number of randomly drawn sample points are needed to have similar properties in regard to the sampling distribution. In addition, the deterministically chosen sample points offer a symmetric set of sampling points that are more statistically meaningful than normally distributed random points.

C.2 Unscented GA Lunar Lander Optimal Control Application

This section summarizes the results obtained by implementing a rudimentary version of the unscented DGX crossover operator and using it to solve the lunar lander problem outlined in Equation (4.3). The same top-level RCGA algorithm from Chapter 4 is used where tournament selection is performed, considering both the parents and offspring, to create the next set of parents. In addition, elitism is enforced from one generation to the next, ensuring that the best solution found is always in the population. In this algorithm, the same cooling schedule that is used for the standard DGX crossover operator is also applied to this new unscented crossover operator. In this unscented crossover operator, 3rd order sigma points are used with $\kappa = 1$ to generate $2n + 1$ candidate offspring solutions from a normal distribution. The n-dimensional normal distribution is centered on a mean that is equal to the fitness-weighted average between the two parent solution vectors. Next, all of the $2n + 1$ candidate offspring vectors are evaluated against the fitness function of interest. From here, a modified truncation selection algorithm is used to quickly find the two best

performing offspring. These offspring are then used as they would be from any of the other real-coded crossover operators considered in Chapter 4.

The specific settings of this unscented RCGA that are used for solving the lunar lander problem are outlined in Table C.1. This algorithm is applied using the exact same parallel island architecture with a round-robin communication network as described in Section 4.5.3. It also has a covariance reset mechanism where the covariance is reset back to the initial value when the average population fitness stagnates and does not change by more than a given reset tolerance from one generation to the next. Lastly, the covariance decay rate is the user-defined parameter α as defined in Equation (4.24).

Parameter	Value
Population Size	100
p_{cross}	0.85
p_{mut}	0.001
Tournament Pool Size	10
Initial Covariance	$\frac{UB_i - LB_i}{2}$
Covariance Reset Tolerance	10^{-6}
Covariance Decay Rate	1.001
Exit Criteria	10^6 Generations

Table C.1. Unscented RCGA parameters

Table C.2 and Figure C.1 illustrate the results obtained from applying this new form of unscented GA to the lunar lander optimal control problem outlined in Equation (4.3). A parallel island version of this unscented RCGA is run on Hamming using 256 processors across eight different compute nodes (32 processors per node). It can be seen that the unscented GA is able to find very accurate solutions to this problem. However, it does take a longer amount of time to run, when compared to the USA-ES solution for the 30-node lunar lander problem in Table 6.25, given that it has a much larger number of function evaluations to execute during each generation to enable the additional fitness-based selection that occurs within the crossover operator.

Method	Migrations 10^6 Gen.	F_{Best}	F_{Best} Error (%)	CPU Time (hours)	Time Steps
Unscented RCGA	256	1.420305	$1.9 \times 10^{-3}\%$	3.7	30

Table C.2. Unscented RCGA lunar lander results

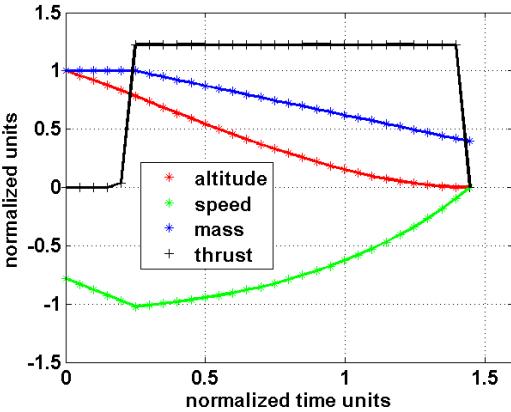


Figure C.1. Unscented RCGA 30-node lunar lander trajectory

As alluded to before, this algorithm could be made to run faster by utilizing a random selection operator to choose the two offspring from the group of $2n + 1$ solution vectors generated by the 3rd order sigma points instead of using a fitness-based truncation selection. In addition, the best solution for the lunar lander problem outlined in Table C.2 and Figure C.1 is actually found in generation 457,497 for this specific run, indicating that 10^6 generations may be more than twice as many generations than what is actually needed for this problem. This means that the solution found using this algorithm was actually found in a little less than half of the time indicated in Table C.2.

While this algorithm does share many similar traits to the USA-ES it also has several differences that may make it very useful for multimodal type problems. First of all, there is one additional level of parallelism that is going on with the parallel unscented RCGAs. In this algorithm, each processor is working with $\frac{N_{pop}}{2}$ n-dimensional normal distributions during each generation that are all potentially spread across the search domain. This is due to the fact that each processor is working with its own population where two parents at a time are paired off to determine the mean value for each of these potentially unique distributions. As an example, the algorithm used in this lunar lander example has 50 sets of parents defining 50 n-dimensional normal distributions per processor. This means that there are 12,800 normal distributions being sampled from during each generation when using 256 processors. Next, there are a couple of additional elements of randomness that are in effect on these unscented RCGAs that don't exist within the construct of a USA-ES.

The first element of randomness arises from the use of the tournament selection method to select the new parent pools for each generation, where only truncation selection is used within the USA-ES. The second element of randomness stems from the idea that two new parents are selected to define each normal distribution for each generation. This makes it so that the location of these distributions can change fairly rapidly from one generation to the next, depending on the location and performance of the chosen parents. Lastly, if a random selection operator were to be used to select the two offspring within the unscented crossover operator, this would further add another random element to the algorithm. This of course could be of benefit on some problems and could degrade algorithm performance on others. Given this, there are a number of further modifications and trades that could be done regarding this idea of an unscented RCGAs.

THIS PAGE INTENTIONALLY LEFT BLANK

List of References

- [1] W. Huang, Y. Nakamori, and S.-Y. Wang, “Forecasting stock market movement direction with support vector machine,” *Computers & Operations Research*, vol. 32, no. 10, pp. 2513–2522, Oct. 2005. Available: <http://linkinghub.elsevier.com/retrieve/pii/S0305054804000681>
- [2] J. Biles, “GenJam: A genetic algorithm for generating jazz solos,” in *Proceedings of the International Computer Music Conference*, 1994, pp. 131–131. Available: <http://igm.rit.edu/~jabics/GenJam94/Paper.html>
- [3] G. A. Dukeman and A. Hill, “Rapid trajectory optimization for the Ares I launch vehicle,” *AIAA Paper*, vol. 6288, 2008. Available: <http://arc.aiaa.org/doi/pdf/10.2514/6.2008-6288>
- [4] J. Nieberding and C. H. Williams, “ADDJUST-A view of the first 25 years,” in *Proceedings of the 50th Joint Propulsion Conference*, Cleveland, OH, 2014, vol. AIAA 3672. Available: <http://ntrs.nasa.gov/search.jsp?R=20150010217>
- [5] J. Zhao, R. Zhou, and X. Jin, “Progress in reentry trajectory planning for hypersonic vehicle,” *Journal of Systems Engineering and Electronics*, vol. 25, no. 4, pp. 627–639, Aug. 2014. Available: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6905957>
- [6] P. F. Gath and A. J. Calise, “Optimization of launch vehicle ascent trajectories with path constraints and coast arcs,” *Journal of Guidance, Control, and Dynamics*, vol. 24, no. 2, pp. 296–304, 2001. Available: <http://arc.aiaa.org/doi/pdf/10.2514/2.4712>
- [7] J. Rea, “Launch vehicle trajectory optimization using a Legendre pseudospectral method,” in *Proceedings of the AIAA Guidance, Navigation and Control Conference*, Austin, TX, 2003, vol. 5640. Available: <http://arc.aiaa.org/doi/pdf/10.2514/6.2003-5640>
- [8] S. C. W. Steele, “Optimal Engine Selection and Trajectory Optimization using Genetic Algorithms for Conceptual Design Optimization of Resuable Launch Vehicles,” Ph.D. dissertation, Virginia Tech, 2015. Available: <https://vttechworks.lib.vt.edu/handle/10919/51771>
- [9] M. Balesdent, N. Bérend, P. Dépincé, and A. Chriette, “A survey of multidisciplinary design optimization methods in launch vehicle design,” *Structural and Multidisciplinary Optimization*, vol. 45, no. 5, pp. 619–642, 2012. Available: <http://link.springer.com/article/10.1007/s00158-011-0701-4>

- [10] J. A. Englander, B. A. Conway, and T. Williams, “Automated mission planning via evolutionary algorithms,” *Journal of Guidance, Control, and Dynamics*, vol. 35, no. 6, pp. 1878–1887, 2012. Available: <http://arc.aiaa.org/doi/pdf/10.2514/1.54101>
- [11] J. A. Englander and M. A. Vavrina, “Multi-Objective Hybrid Optimal Control for Multiple-Flyby Interplanetary Mission Design Using Chemical Propulsion,” in *Proceedings of AAS/AIAA Space Flight Mechanics Meeting*, Williamsburg, VA, 2015. Available: <http://ntrs.nasa.gov/search.jsp?R=20150020820>
- [12] M. P. Ferringer and D. B. Spencer, “Satellite constellation design tradeoffs using multiple-objective evolutionary computation,” *Journal of spacecraft and rockets*, vol. 43, no. 6, pp. 1404–1411, 2006. Available: <http://arc.aiaa.org/doi/pdf/10.2514/1.18788>
- [13] M. P. Ferringer, R. S. Clifton, and T. G. Thompson, “Efficient and accurate evolutionary multi-objective optimization paradigms for satellite constellation design,” *Journal of Spacecraft and Rockets*, vol. 44, no. 3, pp. 682–691, 2007. Available: <http://arc.aiaa.org/doi/pdf/10.2514/1.26747>
- [14] J. R. Wertz, D. F. Everett, and J. J. Puschell, *Space mission engineering: the new SMAD*. Hawthorne, CA: Microcosm Press, 2011.
- [15] N. Bedrossian, S. Bhatt, W. Kang, and I. Ross, “Zero-propellant maneuver guidance,” *IEEE Control Systems Magazine*, vol. 29, no. 5, pp. 53–73, Oct. 2009. Available: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5256357>
- [16] M. Karpenko, S. Bhatt, N. Bedrossian, and I. M. Ross, “Flight implementation of shortest-time maneuvers for imaging satellites,” *Journal of Guidance, Control, and Dynamics*, vol. 37, no. 4, pp. 1069–1079, July 2014. Available: <http://arc.aiaa.org/doi/abs/10.2514/1.62867>
- [17] N. M. Horri, P. Palmer, and S. Hodgart, “Practical implementation of attitude-control algorithms for an underactuated satellite,” *Journal of Guidance, Control, and Dynamics*, vol. 35, no. 1, pp. 40–45, 2012. Available: <http://arc.aiaa.org/doi/abs/10.2514/1.54075>
- [18] C. D. Petersen, F. Leve, M. Flynn, and I. Kolmanovsky, “Recovering linear controllability of an underactuated spacecraft by exploiting solar radiation pressure,” *Journal of Guidance, Control, and Dynamics*, vol. 39, no. 4, pp. 1–12, 2015. Available: <http://arc.aiaa.org/doi/abs/10.2514/1.G001446>
- [19] J. Frank, A. Jonsson, R. Morris, D. E. Smith, and P. Norvig, “Planning and scheduling for fleets of earth observing satellites,” in *Proceedings of the 6th International Symposium on Artificial Intelligence, Robotics, Automation and Space*, 2001. Available: <http://ntrs.nasa.gov/search.jsp?R=20010087327>

- [20] A. Globus, J. Crawford, J. Lohn, and A. Pryor, “Scheduling earth observing satellites with evolutionary algorithms,” in *Proceedings of the International Conference on Space Mission Challenges for Information Technology*, Pasadena, CA, 2003. Available: <http://ntrs.nasa.gov/search.jsp?R=20030062898>
- [21] A. Globus, J. Crawford, J. Lohn, and A. Pryor, “A comparison of techniques for scheduling earth observing satellites,” in *Proceedings of the 16th Innovative Applications of Artificial Intelligence Conference*, San Jose, CA, 2004, pp. 836–843. Available: <http://www.aaai.org/Papers/IAAI/2004/IAAI04-010.pdf>
- [22] L. Barbulescu, A. E. Howe, L. D. Whitley, and M. Roberts, “Understanding algorithm performance on an oversubscribed scheduling application,” *Journal of Artificial Intelligence Research*, pp. 577–615, 2006. Available: <http://www.jair.org/papers/paper2038.html>
- [23] C. M. Chilan and B. A. Conway, “Automated design of multiphase space missions using hybrid optimal control,” *Journal of Guidance, Control, and Dynamics*, vol. 36, no. 5, pp. 1410–1424, Sep. 2013. Available: <http://arc.aiaa.org/doi/abs/10.2514/1.58766>
- [24] M. P. Patel, J. M. DiCocco, T. S. Prince, and T. T. Ng, “Afterbody flow control for low alpha missile maneuvering,” *AIAA*, vol. 3673, 2003. Available: <http://arc.aiaa.org/doi/pdf/10.2514/6.2003-3673>
- [25] C.-K. Ryoo, H. Cho, and M.-J. Tahk, “Optimal guidance laws with terminal impact angle constraint,” *Journal of Guidance, Control, and Dynamics*, vol. 28, no. 4, pp. 724–732, 2005. Available: <http://arc.aiaa.org/doi/abs/10.2514/1.8392>
- [26] L. Walter, G. Schloffel, S. Theodoulis, P. Wernert, E. Kostina, and F. Holzapfel, “Optimal control and numerical optimization for missile interception guidance,” in *Proceedings of European Control Conference*. IEEE, 2014, pp. 1249–1255. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6862503
- [27] A. F. Woolf, “Conventional prompt global strike and long-range ballistic missiles: background and issues,” 2015. Available: <http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA623551>
- [28] J. J. Spravka and T. R. Jorris, “Current Hypersonic and Space Vehicle Flight Test and Instrumentation,” Air Force Test Center, Edwards AFB, CA, Tech. Rep. 412TW-PA-15264, 2015. Available: <http://arc.aiaa.org/doi/pdf/10.2514/6.2015-3224>

- [29] Y. Li, N. Cui, and S. Rong, “Trajectory optimization for hypersonic boost-glide missile considering aeroheating,” *Aircraft Engineering and Aerospace Technology*, vol. 81, no. 1, pp. 3–13, 2009. Available: <http://www.emeraldinsight.com/doi/abs/10.1108/00022660910926854>
- [30] J. T. Betts, “Survey of numerical methods for trajectory optimization,” *Journal of Guidance, Control, and Dynamics*, vol. 21, no. 2, pp. 193–207, Mar. 1998. Available: <http://arc.aiaa.org/doi/abs/10.2514/2.4231>
- [31] I. M. Ross, *A primer on Pontryagin’s principle in optimal control*. San Francisco, CA: Collegiate Publishers, 2009.
- [32] A. V. Rao, “A survey of numerical methods for optimal control,” *Advances in the Astronautical Sciences*, vol. 135, no. 1, pp. 497–528, 2009. Available: <http://vdol.mae.ufl.edu/ConferencePublications/trajectorySurveyAAS.pdf>
- [33] B. A. Conway, “A survey of methods available for the numerical optimization of continuous dynamic systems,” *Journal of Optimization Theory and Applications*, vol. 152, no. 2, pp. 271–306, Feb. 2012. Available: <http://link.springer.com/10.1007/s10957-011-9918-z>
- [34] R. L. Rardin, *Optimization in operations research*. Upper Saddle River, NJ: Prentice Hall, 1998, vol. 166. Available: http://fora.aa.ufl.edu/docs/47/19Feb13/UCC_19Feb13_Undergrad2_EG-ESI%204312.pdf
- [35] C. Geyer, “Introduction to Markov Chain Monte Carlo,” in *Handbook of Markov Chain Monte Carlo*. London: Chapman and Hall, 2011, pp. 3–48. Available: <https://books.google.com/books?hl=en&lr=&id=qfRsAIKZ4rIC&oi=fnd&pg=PA3&dq=Handbook+of+Markov+Chain+Monte+Carlo&ots=Rby5cP-j2T&sig=yqcXPoBSMHAWmqhRr0amBWzZKFw>
- [36] T. Bäck and H.-P. Schwefel, “An overview of evolutionary algorithms for parameter optimization,” *Evolutionary computation*, vol. 1, no. 1, pp. 1–23, 1993. Available: <http://www.mitpressjournals.org/doi/abs/10.1162/evco.1993.1.1.1>
- [37] D. Whitley, “An overview of evolutionary algorithms: practical issues and common pitfalls,” *Information and software technology*, vol. 43, no. 14, pp. 817–831, 2001. Available: <http://www.sciencedirect.com/science/article/pii/S0950584901001884>
- [38] H.-G. Beyer, H.-P. Schwefel, and I. Wegener, “How to analyse evolutionary algorithms,” *Theoretical Computer Science*, vol. 287, no. 1, pp. 101–130, 2002. Available: <http://www.sciencedirect.com/science/article/pii/S0304397502001378>

- [39] J. H. Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. Ann Arbor, MI: University of Michigan Press, 1975. Available: <http://psycnet.apa.org/psycinfo/1975-26618-000>
- [40] I. Rechenberg, “Evolution strategy: optimization of technical systems by means of biological evolution,” *Fromman-Holzboog, Stuttgart*, vol. 104, 1973.
- [41] D. Wierstra, T. Schaul, T. Glasmachers, Y. Sun, J. Peters, and J. Schmidhuber, “Natural evolution strategies,” *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 949–980, 2014. Available: <http://dl.acm.org/citation.cfm?id=2638566>
- [42] E. Cantu-Paz and D. E. Goldberg, “Efficient parallel genetic algorithms: theory and practice,” *Computer Methods in Applied Mechanics and Engineering*, vol. 186, no. 2, pp. 221–238, 2000. Available: <http://www.sciencedirect.com/science/article/pii/S0045782599003850>
- [43] K. Weinert, J. Mehnen, and G. Rudolph, “Dynamic neighborhood structures in parallel evolution strategies,” *Complex Systems*, vol. 13, no. 3, pp. 227–244, 2001. Available: <http://www.complex-systems.com/pdf/13-3-3.pdf>
- [44] M. D. Vose, “What are genetic algorithms? A mathematical perspective,” in *Evolutionary Algorithms*. Springer, 1999, pp. 251–276. Available: http://link.springer.com/chapter/10.1007/978-1-4612-1542-4_14
- [45] D. H. Wolpert and W. G. Macready, “No free lunch theorems for optimization,” *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997. Available: <http://ti.arc.nasa.gov/m/profile/dhw/papers/78.pdf>
- [46] C. McGrath, M. Karpenko, and R. J. Proulx, “Distributed computation for near real-time footprint generation,” in *Proceedings of The 2015 AAS/AIAA Astrodynamics Specialist Conference*, Vail, CO, 2015.
- [47] C. McGrath, M. Karpenko, and R. J. Proulx, “Parallel genetic algorithms for optimal control,” in *Proceedings of The 2016 AAS/AIAA Spaceflight Mechanics Conference*, Napa, CA, 2016.
- [48] C. McGrath, M. Karpenko, R. J. Proulx, and I. M. Ross, “Unscented evolution strategies for solving trajectory optimization problems,” in *Proceedings of The 2016 AAS/AIAA Spaceflight Mechanics Conference*, Napa, CA, 2016.
- [49] N. Hansen, D. V. Arnold, and A. Auger, “Evolution strategies,” in *Handbook of Computational Intelligence*. Springer, 2015, pp. 871–898. Available: http://link.springer.com/chapter/10.1007/978-3-662-43505-2_44

- [50] D. Wierstra, T. Schaul, J. Peters, and J. Schmidhuber, “Natural evolution strategies,” in *Proceedings of the IEEE World Congress on Evolutionary Computation*. IEEE, 2008, pp. 3381–3387. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4631255
- [51] H. Sutter, “The free lunch is over: A fundamental turn toward concurrency in software,” *Dr. Dobb’s journal*, vol. 30, no. 3, pp. 202–210, 2005. Available: <http://mondrian.die.udec.cl/~mmedina/Clases/ProgPar/Sutter%20-%20The%20Free%20Lunch%20is%20Over.pdf>
- [52] J. Larus, “Spending Moore’s dividend,” *Communications of the ACM*, vol. 52, no. 5, pp. 62–69, 2009. Available: <http://dl.acm.org/citation.cfm?id=1506425>
- [53] K. Olukotun and L. Hammond, “The future of microprocessors,” *Queue*, vol. 3, no. 7, pp. 26–29, 2005. Available: <http://dl.acm.org/citation.cfm?id=1095418>
- [54] P. Grogono and B. Shearing, “Concurrent software engineering: Preparing for paradigm shift,” in *Proceedings of the 2008 C 3 S 2 E conference*. ACM, 2008, pp. 99–108. Available: <http://dl.acm.org/citation.cfm?id=1370270>
- [55] H. Sutter and J. Larus, “Software and the concurrency revolution,” *Queue*, vol. 3, no. 7, pp. 54–62, 2005. Available: <http://dl.acm.org/citation.cfm?id=1095421>
- [56] T. Rauber and G. Rünger, *Parallel programming: For multicore and cluster systems*. Springer Science & Business Media, 2013. Available: https://books.google.com/books?hl=en&lr=&id=UbpAAAAQBAJ&oi=fnd&pg=PR5&dq=rauber+and+runger&ots=9YJCcFnLxC&sig=Y_EsCDeRfX-zPU2CljcuAkuRNEg
- [57] S. Larsen, R. Rabbah, and S. Amarasinghe, “Exploiting vector parallelism in software pipelined loops,” in *Proceedings of the 38th annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 2005, pp. 119–129. Available: <http://dl.acm.org/citation.cfm?id=1100544>
- [58] B. Gaster, L. Howes, D. R. Kaeli, P. Mistry, and D. Schaa, *Heterogeneous Computing with OpenCL: Revised OpenCL 1.2 Edition*. Newnes, Dec. 2012.
- [59] J. Haferman, “Hamming architecture - high performance computing - NPS wiki,” Apr. 2016. Available: <https://wiki.nps.edu/display/HPC/Hamming+Architecture>
- [60] J. Monaco, D. Ward, J. Schierman, and J. Hull, “A reconfigurable guidance approach for reusable launch vehicles.” American Institute of Aeronautics and Astronautics, Aug. 2001. Available: <http://arc.aiaa.org/doi/abs/10.2514/6.2001-4429>

- [61] D. Rasky, R. B. Pittman, and M. Newfield, “The reusable launch vehicle challenge,” in *Proceedings of AIAA Space 2006*. AIAA, Sep. 2006. Available: <http://arc.aiaa.org/doi/abs/10.2514/6.2006-7208>
- [62] K. P. Bollino, “High-fidelity real-time trajectory optimization for reusable launch vehicles,” Ph.D. dissertation, Naval Postgraduate School, Monterey, CA, 2006. Available: <http://calhoun.nps.edu/handle/10945/10056>
- [63] Z. Shen and P. Lu, “Onboard generation of three-dimensional constrained entry trajectories,” *Journal of Guidance, Control, and Dynamics*, vol. 26, no. 1, pp. 111–121, Jan. 2003. Available: <http://arc.aiaa.org/doi/abs/10.2514/2.5021>
- [64] N. D. Moshman and R. J. Proulx, “Range improvements in gliding reentry vehicles from thrust capability,” *Journal of Spacecraft and Rockets*, vol. 51, no. 5, pp. 1681–1694, Sep. 2014. Available: <http://arc.aiaa.org/doi/abs/10.2514/1.A32764>
- [65] J. Benito and K. D. Mease, “Reachable and controllable sets for planetary entry and landing,” *Journal of Guidance, Control, and Dynamics*, vol. 33, no. 3, pp. 641–654, May 2010. Available: <http://arc.aiaa.org/doi/abs/10.2514/1.47577>
- [66] K. P. Bollino and Ross, I. M., “A pseudospectral feedback method for real-time optimal guidance of reentry vehicles,” in *Proceedings of American Control Conference*. IEEE, 2007, pp. 3861–3867. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4282500
- [67] MATLAB, *MATLAB documentation (R2015a): Commonly used startup options*. Natick, Massachusetts: The MathWorks Inc., 2015. Available: http://www.mathworks.com/help/matlab/matlab_env/commonly-used-startup-options.html
- [68] K. A. De Jong, “Analysis of the behavior of a class of genetic adaptive systems,” Ph.D Dissertation, Dept. Comput. Sci. Univ. Michigan, Ann Arbor, MI, 1975. Available: <http://deepblue.lib.umich.edu/handle/2027.42/4507>
- [69] L. Davis, “Adapting operator probabilities in genetic algorithms,” in *Proceedings of the 3rd International Conference on Genetic Algorithms*, 1989, pp. 61–69. Available: <http://ci.nii.ac.jp/naid/10000130324/>
- [70] C. B. Lucasius and G. Kateman, “Application of genetic algorithms in chemometrics,” in *Proceedings of the 3rd International Conference on Genetic Algorithms*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1989, pp. 170–176. Available: <http://dl.acm.org/citation.cfm?id=93126.93184>
- [71] H. Yu and J. A. Mulder, “Arrival trajectory optimization for passenger aircraft using genetic algorithms.” AIAA, Sep. 2011. Available: <http://arc.aiaa.org/doi/abs/10.2514/6.2011-6804>

- [72] R. Farmani, J. A. Wright, D. A. Savic, and G. A. Walters, “Self-Adaptive Fitness Formulation for Evolutionary Constrained Optimization of Water Systems,” *Journal of Computing in Civil Engineering*, vol. 19, no. 2, pp. 212–216, Apr. 2005.
- [73] C. R. Reeves and J. E. Rowe, *Genetic algorithms: principles and perspectives*. Kluwer Academic Publishers, 2003.
- [74] V. Toğan and A. T. Daloğlu, “An improved genetic algorithm with initial population strategy and self-adaptive member grouping,” *Computers & Structures*, vol. 86, no. 11, pp. 1204–1218, 2008. Available: <http://www.sciencedirect.com/science/article/pii/S004579490700301X>
- [75] D. E. Goldberg and K. Deb, “A comparative analysis of selection schemes used in genetic algorithms,” *Foundations of genetic algorithms*, vol. 1, pp. 69–93, 1991. Available: <https://books.google.com/books?hl=en&lr=&id=3TqeBQAAQBAJ&oi=fnd&pg=PA69&dq=an+investigation+of+niching+and+species+formation+in+genetic+algorithms&ots=Y4468pa-jE&sig=VC-f85X1GFgHtcVDE0Z4QUnHEGM>
- [76] T. Bickle and L. Thiele, “A comparison of selection schemes used in evolutionary algorithms,” *Evolutionary Computation*, vol. 4, pp. 361–394, 1997.
- [77] J. E. Baker, “Adaptive selection methods for genetic algorithms,” in *Proceedings of an International Conference on Genetic Algorithms and their applications*. Hillsdale, New Jersey, 1985, pp. 101–111. Available: <https://books.google.com/books?hl=en&lr=&id=II17AgAAQBAJ&oi=fnd&pg=PA101&dq=ranking+selection+in+genetic+algorithms&ots=0Ko2a6R97u&sig=du4VvYySNbmhxQNNcKa2x3A6qMU>
- [78] J. E. Baker, “Reducing bias and inefficiency in the selection algorithm,” in *Proceedings of the second international conference on genetic algorithms*, 1987, pp. 14–21. Available: https://books.google.com/books?hl=en&lr=&id=MYJ_AAAQBAJ&oi=fnd&pg=PA14&ots=XvrMsl7DDC&sig=mM4B1TKbiQ_6oa0xT45DIGXyWlc
- [79] T. Bäck, F. Hoffmeister, and H.-P. Schwefel, “A survey of evolution strategies,” in *Proceedings of The 4th International Conference on Genetic Algorithms*, 1991, pp. 2–9. Available: <http://bioserver.cpgei.ct.utfpr.edu.br/disciplinas/ce-antigo/zips/back91survey.pdf>
- [80] H. Mühlenbein and D. Schlierkamp-Voosen, “Predictive models for the breeder genetic algorithm I. continuous parameter optimization,” *Evolutionary Computation*, vol. 1, no. 1, pp. 25–49, Mar. 1993.

- [81] G. Rudolph, “Convergence analysis of canonical genetic algorithms,” *Neural Networks, IEEE Transactions on*, vol. 5, no. 1, pp. 96–101, 1994. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=265964
- [82] R. A. Caruana and J. D. Schaffer, “Representation and hidden bias: gray vs. binary coding for genetic algorithms,” in *Proceedings of the Fifth International Conference on Machine Learning Ann Arbor, Mich.*, 1988, pp. 153–161. Available: <http://ci.nii.ac.jp/naid/10003271733/>
- [83] D. Whitley, “A free lunch proof for gray versus binary encodings,” in *Proceedings of the Genetic and Evolutionary Computation Conference*. Citeseer, 1999, vol. 1, pp. 726–733. Available: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.42.2555&rep=rep1&type=pdf>
- [84] G. Syswerda, “Uniform crossover in genetic algorithms,” *Proceedings of the 3rd International Conference on Genetic Algorithms*, pp. 2–9, 1989. Available: <http://www.citeulike.org/group/314/article/904846>
- [85] K. A. De Jong and W. M. Spears, “An analysis of the interacting roles of population size and crossover in genetic algorithms,” in *Parallel problem solving from nature*. Springer, 1991, pp. 38–47. Available: <http://link.springer.com/chapter/10.1007/BFb0029729>
- [86] M. Srinivas and L. M. Patnaik, “Adaptive probabilities of crossover and mutation in genetic algorithms,” *Transactions on Systems, Man and Cybernetics, IEEE*, vol. 24, no. 4, pp. 656–667, 1994. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=286385
- [87] J. Dutta, K. Deb, R. Tulshyan, and R. Arora, “Approximate KKT points and a proximity measure for termination,” *Journal of Global Optimization*, vol. 56, no. 4, pp. 1463–1499, Aug. 2013. Available: <http://link.springer.com/10.1007/s10898-012-9920-5>
- [88] R. Tulshyan, R. Arora, K. Deb, and J. Dutta, “Investigating EA solutions for approximate KKT conditions in smooth problems,” in *Proceedings of the 12th annual conference on Genetic and evolutionary computation*. ACM, 2010, pp. 689–696. Available: <http://dl.acm.org/citation.cfm?id=1830609>
- [89] H. I. Calvete, C. Galé, and P. M. Mateo, “A new approach for solving linear bilevel problems using genetic algorithms,” *European Journal of Operational Research*, vol. 188, no. 1, pp. 14–28, July 2008. Available: <http://linkinghub.elsevier.com/retrieve/pii/S0377221707003773>

- [90] A. Bertoni and M. Dorigo, “Implicit parallelism in genetic algorithms,” *Artificial Intelligence*, vol. 61, no. 2, pp. 307–314, 1993. Available: <http://www.sciencedirect.com/science/article/pii/000437029390071I>
- [91] D. E. Goldberg, “Genetic algorithms and Walsh functions-Part II: Deception and its analysis,” *Complex Systems*, vol. 3, pp. 153–171, 1989. Available: <http://ci.nii.ac.jp/naid/10000080386/>
- [92] P. M. M. Tomassini, Leonardo Collard, “A Study of Fitness Distance Correlation as a Difficulty Measure in Genetic Programming,” *Evolutionary Computation*, vol. 13, no. 2, pp. 213–239, 2005. Available: <http://libproxy.nps.edu/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=cph&AN=17228510&site=ebook-live&scope=site>
- [93] T. Jones, S. Forrest, and others, “Fitness distance correlation as a measure of problem difficulty for genetic algorithms.” in *ICGA*, 1995, vol. 95, pp. 184–192. Available: https://www.researchgate.net/profile/Terry_Jones10/publication/2608247_Fitness_Distance_Correlation_as_a_Measure_of_Problem_Difficulty_for_Genetic_Algorithms/links/0deec53146e02778cd000000.pdf
- [94] D. E. Goldberg, *Genetic algorithms in search optimization and machine learning*. Addison-wesley Reading Menlo Park, 1989, vol. 412. Available: <https://pdfs.semanticscholar.org/146b/b2ea1fbdd86f81cd0dae7d3fd63decac9f5c.pdf>
- [95] K. Deb and D. E. Goldberg, “Sufficient conditions for deceptive and easy binary functions,” *Annals of mathematics and Artificial Intelligence*, vol. 10, no. 4, pp. 385–408, 1994. Available: <http://link.springer.com/article/10.1007/BF01531277>
- [96] S. Forrest and M. Mitchell, “What makes a problem hard for a genetic algorithm? Some anomalous results and their explanation,” *Machine Learning*, vol. 13, no. 2-3, pp. 285–319, Nov. 1993. Available: <http://link.springer.com/article/10.1007/BF00993046>
- [97] D. Whitley, “Fundamental principles of deception,” *Foundations of Genetic Algorithms 1991 (FOGA 1)*, vol. 1, p. 221, 2014. Available: https://books.google.com/books?hl=en&lr=&id=3TqeBQAAQBAJ&oi=fnd&pg=PA221&dq=fundamental+principles+of+deception+in+genetic+search&ots=Y4468oe4jG&sig=etyoSdO-_j-nITm0mua93nfNaQY
- [98] D. E. Goldberg, K. Deb, and J. rey Horn, “Massive multimodality, deception, and genetic algorithms,” *Urbana*, vol. 51, p. 61801, 1992. Available: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.48.8442&rep=rep1&type=pdf>

- [99] D. E. Goldberg, K. Deb, H. Kargupta, and G. Harik, “Rapid, accurate optimization of difficult problems using messy genetic algorithms,” in *Proceedings of the Fifth International Conference on Genetic Algorithms*. Urbana, USA: Proceedings of the Fifth International Conference on Genetic Algorithms,, 1993, pp. 59–64. Available: <http://repository.ias.ac.in/81677/1/110-a.pdf>
- [100] A. Agapie, “Genetic algorithms: Minimal conditions for convergence,” in *Artificial Evolution*. Springer, 1998, pp. 181–193. Available: <http://link.springer.com/chapter/10.1007/BFb0026600>
- [101] H. Aytug, S. Bhattacharrya, and G. J. Koehler, “A Markov chain analysis of genetic algorithms with power of 2 cardinality alphabets,” *European Journal of Operational Research*, vol. 96, no. 1, pp. 195–201, 1997. Available: <http://www.sciencedirect.com/science/article/pii/S037722179600121X>
- [102] H. Aytug and G. J. Koehler, “Stopping criteria for finite length genetic algorithms,” *INFORMS Journal on Computing*, vol. 8, no. 2, pp. 183–191, 1996. Available: <http://pubsonline.informs.org/doi/abs/10.1287/ijoc.8.2.183>
- [103] T. E. Davis and J. C. Principe, “A Markov chain framework for the simple genetic algorithm,” *Evolutionary computation*, vol. 1, no. 3, pp. 269–288, 1993. Available: <http://www.mitpressjournals.org/doi/abs/10.1162/evco.1993.1.3.269>
- [104] H. Dawid, “A Markov chain analysis of genetic algorithms with a state dependent fitness function,” *Complex Systems*, vol. 8, no. 6, pp. 407–418, 1994. Available: <http://www.complex-systems.com/pdf/08-6-2.pdf>
- [105] A. E. Eiben, E. H. Aarts, and K. M. Van Hee, “Global convergence of genetic algorithms: A Markov chain analysis,” in *Parallel problem solving from nature*. Springer, 1991, pp. 3–12. Available: <http://link.springer.com/chapter/10.1007/BFb0029725>
- [106] D. Greenhalgh, Stephen, “Convergence criteria for genetic algorithms,” *SIAM Journal on Computing*, vol. 30, no. 1, p. 269, Aug. 2000. Available: <http://libproxy.nps.edu/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=cph&AN=10475683&site=ehost-live&scope=site>
- [107] G. Guo and S. Yu, “Using the Markov chain of the best individual to analyze convergence of genetic algorithms,” in *IEEE, Proceedings of the Third World Congress on Intelligent Control and Automation*, 2000, vol. 1, p. 28.
- [108] Y. Leung, Y. Gao, and Z.-B. Xu, “Degree of population diversity-a perspective on premature convergence in genetic algorithms and its markov chain analysis,” *IEEE Transactions on Neural Networks*, vol. 8, no. 5, pp. 1165–1176, 1997. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=623217

- [109] A. E. Nix and M. D. Vose, “Modeling genetic algorithms with Markov chains,” *Annals of mathematics and artificial intelligence*, vol. 5, no. 1, pp. 79–88, 1992. Available: <http://link.springer.com/article/10.1007/BF01530781>
- [110] G. Rudolph, “Finite Markov chain results in evolutionary computation: A tour d’horizon,” 1998. Available: <http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.41.3475>
- [111] W. M. Spears and K. A. De Jong, “Analyzing GAs using Markov models with semantically ordered and lumped states.” in *FOGA*, 1996, pp. 85–100. Available: <http://www.swarmotics.com/uploads/foga96.pdf>
- [112] P. Vitányi, “Genetic fitness optimization using rapidly mixing Markov chains,” in *Algorithmic Learning Theory*. Springer, 1996, pp. 67–82. Available: http://link.springer.com/chapter/10.1007/3-540-61863-5_35
- [113] M. D. Vose and G. E. Liepins, “Punctuated equilibria in genetic search,” *Complex systems*, vol. 5, pp. 31–44, 1991. Available: <http://www.complex-systems.com/pdf/05-1-4.pdf>
- [114] K. A. De Jong, W. M. Spears, and D. F. Gordon, “Using Markov chains to analyze GAFOs,” *Foundations of genetic algorithms 3*, vol. 3, pp. 115–137, 1995. Available: https://books.google.com/books?hl=en&lr=&id=HlGeBQAAQBAJ&oi=fnd&pg=PA115&dq=using+markov+chains+to+to+analyze+gafos&ots=rovrobNEt&sig=N2kIow7w3SRj_nce9Q-0WmcBKXQ
- [115] A. Prügel-Bennett and A. Rogers, “Modelling genetic algorithm dynamics,” in *Theoretical Aspects of Evolutionary Computing*. Springer, 2001, pp. 59–85. Available: http://link.springer.com/chapter/10.1007/978-3-662-04448-3_4
- [116] J. L. Shapiro, “Statistical mechanics theory of genetic algorithms,” in *Theoretical Aspects of Evolutionary Computing*. Springer, 2001, pp. 87–108. Available: http://link.springer.com/chapter/10.1007/978-3-662-04448-3_5
- [117] I. Rojas, J. González, H. Pomares, J. J. Merelo, P. A. Castillo, and G. Romero, “Statistical analysis of the main parameters involved in the design of a genetic algorithm,” *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 32, no. 1, pp. 31–37, 2002. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1009128
- [118] J. J. Grefenstette, “Optimization of control parameters for genetic algorithms,” *IEEE Transactions on Systems, Man and Cybernetics*, vol. 16, no. 1, pp. 122–128, 1986. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4075583

- [119] D. Thierens, “Adaptive mutation rate control schemes in genetic algorithms,” in *Proceedings of the 2002 Congress on Evolutionary Computation*. IEEE, 2002, vol. 1, pp. 980–985. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1007058
- [120] J. T. Alander, “On optimal population size of genetic algorithms,” in *Proceedings Computer Systems and Software Engineering*, 1992, pp. 65–70. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=218485
- [121] D. E. Goldberg, *Optimal initial population size for binary-coded genetic algorithms*. Clearinghouse for Genetic Algorithms, Department of Engineering Mechanics, University of Alabama, 1985.
- [122] D. E. Goldberg, “Sizing populations for serial and parallel genetic algorithms,” in *Proceedings of the 3rd International Conference on Genetic Algorithms*. Morgan Kaufmann Publishers Inc., 1989, pp. 70–79. Available: <http://dl.acm.org/citation.cfm?id=657266>
- [123] D. E. Goldberg, K. Deb, and J. H. Clark, “Genetic algorithms, noise, and the sizing of populations,” *Complex Systems*, vol. 6, pp. 333–362, 1991. Available: <http://repository.ias.ac.in/82725/>
- [124] G. Harik, E. Cantú-Paz, D. E. Goldberg, and B. L. Miller, “The gambler’s ruin problem, genetic algorithms, and the sizing of populations,” *Evolutionary Computation*, vol. 7, no. 3, pp. 231–253, 1999. Available: <http://www.mitpressjournals.org/doi/abs/10.1162/evco.1999.7.3.231>
- [125] W. G. Macready and D. H. Wolpert, “Bandit problems and the exploration/exploitation tradeoff,” *IEEE Transactions on Evolutionary Computation*, vol. 2, no. 1, pp. 2–22, 1998. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=728210
- [126] Y. R. Tsoy, “The influence of population size and search time limit on genetic algorithm,” in *Proceedings KORUS 2003.*, 2003, vol. 3, pp. 181–187. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1222860
- [127] D. E. Goldberg, B. Korb, and K. Deb, “Messy genetic algorithms: Motivation, analysis, and first results,” *Complex systems*, vol. 3, no. 5, pp. 493–530, 1989. Available: <http://repository.ias.ac.in/82726/>
- [128] J. Andre, P. Siarry, and T. Dognon, “An improvement of the standard genetic algorithm fighting premature convergence in continuous optimization,” *Advances in engineering software*, vol. 32, no. 1, pp. 49–60, 2001. Available: <http://www.sciencedirect.com/science/article/pii/S0965997800000703>

- [129] D. E. Goldberg and J. Richardson, “Genetic algorithms with sharing for multimodal function optimization,” in *Genetic algorithms and their applications: Proceedings of the Second International Conference on Genetic Algorithms*. Hillsdale, NJ: Lawrence Erlbaum, 1987, pp. 41–49. Available: https://books.google.com/books?hl=en&lr=&id=MYJ_AAAAQBAJ&oi=fnd&pg=PA41&ots=XvrGtn1DGy&sig=n1pahjWubikpWGVRlx3mltsdZ60
- [130] K. Deb and D. E. Goldberg, “An investigation of niche and species formation in genetic function optimization,” in *Proceedings of the 3rd international conference on genetic algorithms*. Morgan Kaufmann Publishers Inc., 1989, pp. 42–50. Available: <http://dl.acm.org/citation.cfm?id=657099>
- [131] B. L. Miller and D. E. Goldberg, “Genetic algorithms, selection schemes, and the varying effects of noise,” *Evolutionary Computation*, vol. 4, no. 2, pp. 113–131, 1996. Available: <http://www.mitpressjournals.org/doi/abs/10.1162/evco.1996.4.2.113>
- [132] D. K. He and F. L. Wang, “Improving the global convergence of the genetic algorithm,” *Dongbei Daxue Xuebao/Journal of Northeastern University*, vol. 24, no. 6, pp. 511–514, 2003.
- [133] D. E. Goldberg, “Genetic algorithms in search, optimization, and machine learning,” University of Illinois at Urbana-Champaign, University of Illinois at Urbana-Champaign, Technical Report 90001, 1989.
- [134] F. Herrera, M. Lozano, and J. L. Verdegay, “Tackling real-coded genetic algorithms: Operators and tools for behavioural analysis,” *Artificial intelligence review*, vol. 12, no. 4, pp. 265–319, 1998. Available: <http://link.springer.com/article/10.1023/A:1006504901164>
- [135] A. H. Wright, “Genetic algorithms for real parameter optimization,” *Foundations of genetic algorithms*, vol. 1, pp. 205–218, 1991. Available: <https://books.google.com/books?hl=en&lr=&id=3TqeBQAAQBAJ&oi=fnd&pg=PA205&dq=genetic+algorithm+population+size&ots=Y44b6tf0pB&sig=64sOg9wXMz9FkdSDkez1XOwvqJM>
- [136] C. Z. Janikow and Z. Michalewicz, “An experimental comparison of binary and floating point representations in genetic algorithms.” in *ICGA*, 1991, pp. 31–36. Available: <http://www.cs.umsl.edu/~janikow/publications/1991/GAbin/text.pdf>
- [137] D. E. Goldberg, “Real-coded genetic algorithms, virtual alphabets, and blocking,” *Urbana*, vol. 51, p. 61801, 1990. Available: <http://illigal.org/wp-content/uploads/illigal/pub/papers/IlliGALs/90001.pdf>

- [138] N. J. Radcliffe, “Equivalence class analysis of genetic algorithms,” *Complex systems*, vol. 5, no. 2, pp. 183–205, 1991. Available: <http://www-lsic.univ-littoral.fr/~verel/supports/articles/radcliffe91equivalence.pdf>
- [139] L. J. Eshelman and J. D. Schaffer, “Real-coded genetic algorithms and interval-schemata,” *Foundations of Genetic Algorithms*, vol. 2, pp. 187–202, 1993. Available: <http://ci.nii.ac.jp/naid/10011199238/>
- [140] L. J. Eshelman, “Crossover operator biases: Exploiting the population distribution,” in *Proceedings of the 7th International Conference on Genetic Algorithms*, 1997, 1997. Available: <http://ci.nii.ac.jp/naid/80009812991/>
- [141] Z. Michalewicz, *Genetic algorithms+ data structures= evolution programs*. Springer Science & Business Media, 2013.
- [142] Z. Michalewicz, G. Nazhiyath, and M. Michalewicz, “A note on usefulness of geometrical crossover for numerical optimization problems,” *Evolutionary Programming*, vol. 5, no. 1, pp. 305–312, 1996. Available: <http://cs.adelaide.edu.au/users/zbyszek/Papers/p25.pdf>
- [143] S. Tsutsui, M. Yamamura, and T. Higuchi, “Multi-parent recombination with simplex crossover in real coded genetic algorithms,” in *Proceedings of the genetic and evolutionary computation conference*, 1999, vol. 1, pp. 657–664. Available: <http://www2.hannan-u.ac.jp/~tsutsui/ps/icga99.pdf>
- [144] H. Kita, I. Ono, and S. Kobayashi, “Theoretical analysis of the unimodal normal distribution crossover for real-coded genetic algorithms,” in *Proceedings of IEEE World Congress on Computational Intelligence*. IEEE, 1998, pp. 529–534. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=700084
- [145] K. Deb and R. B. Agrawal, “Simulated binary crossover for continuous search space,” *Complex Systems*, vol. 9, no. 3, pp. 1–15, 1994. Available: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.26.8485&rep=rep1&type=pdf>
- [146] H.-G. Beyer and K. Deb, “On self-adaptive features in real-parameter evolutionary algorithms,” *IEEE Transactions on Evolutionary Computation*, vol. 5, no. 3, pp. 250–270, 2001. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=930314
- [147] H.-M. Voigt, H. Mühlenbein, and D. Cvetkovic, “Fuzzy recombination for the breeder genetic algorithm,” in *Proceedings of the 6th International Conference on Genetic Algorithms*. Citeseer, 1995. Available: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.8.3971>

- [148] N. J. Radcliffe, “Forma analysis and random respectful recombination.” in *Proceedings of the International Conference on Genetic Algorithms*, 1991, vol. 91, pp. 222–229. Available: <http://stochasticsolutions.com/pdf/icga91.pdf>
- [149] X. Qi and F. Palmieri, “Theoretical analysis of evolutionary algorithms with an infinite population size in continuous space. Part II: Analysis of the diversification role of crossover,” *IEEE Transactions on Neural Networks*, vol. 5, no. 1, pp. 120–129, 1994. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=265966
- [150] X. Qi and F. Palmieri, “Theoretical analysis of evolutionary algorithms with an infinite population size in continuous space. Part I: Basic properties of selection and mutation,” *IEEE Transactions on Neural Networks*, vol. 5, no. 1, pp. 102–119, 1994. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=265965
- [151] H.-G. Beyer, *The theory of evolution strategies*. Springer Science & Business Media, 2013. Available: https://books.google.com/books?hl=en&lr=&id=f9eoCAAAQBAJ&oi=fnd&pg=PA1&dq=the+theory+of+evolution+strategies&ots=L0FJ4ZtMeY&sig=_SZuI7H9NJzOcByCz72VwH6rbxo
- [152] M. Wahib, A. Munawar, M. Munetomo, and K. Akama, “Optimization of parallel genetic algorithms for nVidia GPUs,” in *Proceedings of the 2011 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2011, pp. 803–811. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5949701
- [153] E. Alba and J. M. Troya, “A survey of parallel distributed genetic algorithms,” *Complexity*, vol. 4, no. 4, pp. 31–52, 1999. Available: http://neo.lcc.uma.es/Articles/albatroyaxx_2.pdf
- [154] P. Pospichal, J. Jaros, and J. Schwarz, “Parallel Genetic Algorithm on the CUDA Architecture,” in *Proceedings of the 2010 International Conference on Applications of Evolutionary Computation - Volume Part I* (EvoApplicatons’10). Berlin, Heidelberg: Springer-Verlag, 2010, pp. 442–451. Available: http://dx.doi.org/10.1007/978-3-642-12239-2_46
- [155] M. Rebaudengo and M. S. Reorda, “An experimental analysis of the effects of migration in parallel genetic algorithms,” in *Proceedings Euromicro Workshop on Parallel and Distributed Processing*. IEEE, 1993, pp. 232–238. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=336398
- [156] E. Cantu-Paz, “Designing efficient and accurate parallel genetic algorithms,” Illinois Genetic Algorithms Laboratory., UIUC, USA, Technical Report 99017, 1999. Available: <http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.44.6123>

- [157] D. Whitley, S. Rana, and R. B. Heckendorn, “The island model genetic algorithm: on separability, population size and convergence,” *Journal of Computing and Information Technology*, vol. 7, pp. 33–48, 1999. Available: <http://neo.lcc.uma.es/Articles/WRH98.pdf>
- [158] E. Cantu-Paz, “Migration policies, selection pressure, and parallel evolutionary algorithms,” *Journal of Heuristics*, vol. 7, no. 4, pp. 311–334, 2001.
- [159] Z. Skolicki and K. De Jong, “The influence of migration sizes and intervals on island models,” in *Proceedings of The 7th Annual Conference on Genetic and Evolutionary Computation*. ACM, 2005, pp. 1295–1302. Available: <http://dl.acm.org/citation.cfm?id=1068219>
- [160] V. S. Gordon and D. Whitley, “Serial and parallel genetic algorithms as function optimizers,” in *Proceedings of International Conference on Genetic Algorithms*, 1993, pp. 177–183. Available: <http://www.cs.colostate.edu/pubserv/pubs/Gordon-TechReports-Reports-1993-tr-114.pdf>
- [161] M. Nowostawski and R. Poli, “Parallel genetic algorithm taxonomy,” in *Proceedings of 3rd International Conference on Knowledge-Based Intelligent Information Engineering Systems*. IEEE, 1999, pp. 88–92. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=820127
- [162] Z. Michalewicz, J. B. Krawczyk, M. Kazemi, and C. Z. Janikow, “Genetic algorithms and optimal control problems,” in *Proceedings of the 29th IEEE Conference on Decision and Control*, 1990, pp. 1664–1666. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=203904
- [163] Z. Michalewicz, C. Z. Janikow, and J. B. Krawczyk, “A modified genetic algorithm for optimal control problems,” *Computers & Mathematics with Applications*, vol. 23, no. 12, pp. 83–94, 1992. Available: <http://www.sciencedirect.com/science/article/pii/089812219290094X>
- [164] H. Seywald and R. Kumar, “Genetic algorithms and equality constrained optimization problems,” in *Proceedings of AIAA Guidance, Navigation and Control Conference*, Baltimore, MD, 1995. Available: <http://arc.aiaa.org/doi/abs/10.2514/6.1995-3225>
- [165] A. Wuerl, T. Crain, and E. Braden, “Genetic algorithm and calculus of variations-based trajectory optimization technique,” *Journal of Spacecraft and Rockets*, vol. 40, no. 6, pp. 882–888, 2003. Available: <http://arc.aiaa.org/doi/abs/10.2514/2.7053>

- [166] N. Yokoyama and S. Suzuki, “Modified genetic algorithm for constrained trajectory optimization,” *Journal of Guidance, Control, and Dynamics*, vol. 28, no. 1, pp. 139–144, 2005. Available: <http://arc.aiaa.org/doi/pdf/10.2514/1.3042>
- [167] K. Subbarao and B. M. Shippey, “Hybrid genetic algorithm collocation method for trajectory optimization,” *Journal of Guidance, Control, and Dynamics*, vol. 32, no. 4, pp. 1396–1403, 2009. Available: <http://arc.aiaa.org/doi/pdf/10.2514/1.41449>
- [168] J. Sha and M. Xu, “Applying hybrid genetic algorithm to constrained trajectory optimization,” in *2011 International Conference on Electronic and Mechanical Engineering and Information Technology (EMEIT)*, Aug. 2011, vol. 7, pp. 3792–3795.
- [169] M. A. Vavrina and K. C. Howell, “Global low-thrust trajectory optimization through hybridization of a genetic algorithm and a direct method,” in *AIAA/AAS Astrodynamics Specialist Conference*, AIAA, 2008, vol. 6614, p. 129143. Available: <http://arc.aiaa.org/doi/pdf/10.2514/6.2008-6614>
- [170] S. Wagner, B. Kaplinger, and B. Wie, “GPU accelerated genetic algorithm for multiple gravity-assist and impulsive V maneuvers,” in *AIAA/AAS Guidance Navigation and Control Conference*, Aug. 2012. Available: <http://arc.aiaa.org/doi/abs/10.2514/6.2012-4592>
- [171] F. Herrera, M. Lozano, and A. M. Sanchez, “A taxonomy for the crossover operator for real-coded genetic algorithms: an experimental study,” *International Journal of Intelligent Systems*, vol. 18, no. 3, pp. 309–338, 2003. Available: <http://onlinelibrary.wiley.com/doi/10.1002/int.10091/abstract>
- [172] G. Di Pillo and L. Grippo, “Exact penalty functions in constrained optimization,” *SIAM Journal on control and optimization*, vol. 27, no. 6, pp. 1333–1360, 1989. Available: <http://pubs.siam.org/doi/abs/10.1137/0327068>
- [173] Z. Meng, Q. Hu, C. Dang, and X. Yang, “An objective penalty function method for nonlinear programming,” *Applied mathematics letters*, vol. 17, no. 6, pp. 683–689, 2004. Available: <http://www.sciencedirect.com/science/article/pii/S089396590490105X>
- [174] C. A. Floudas and P. M. Pardalos, *A Collection of Test Problems for Constrained Global Optimization Algorithms*. Springer Science & Business Media, Sep. 1990.
- [175] A. B. Hadj-Alouane and J. C. Bean, “A genetic algorithm for the multi-purpose integer program,” *Operations Research*, vol. 45, no. 1, p. 92, Feb. 1997. Available: <http://libproxy.nps.edu/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=bth&AN=9709064064&site=ehost-live&scope=site>

- [176] H. Seywald, R. R. Kumar, and S. M. Deshpande, “Genetic algorithm approach for optimal control problems with linearly appearing controls,” *Journal of Guidance, Control, and Dynamics*, vol. 18, no. 1, pp. 177–182, 1995. Available: <http://arc.aiaa.org/doi/pdf/10.2514/3.56673>
- [177] Z. Michalewicz, “A survey of constraint handling techniques in evolutionary computation methods.” *Evolutionary Programming*, vol. 4, pp. 135–155, 1995. Available: <https://books.google.com/books?hl=en&lr=&id=ipiiDmTrJeAC&oi=fnd&pg=PA135&dq=A+survey+of+evolution+strategies+michalewicz&ots=sptu4NA9Ws&sig=77o-lyOvOsXgjFaPoowgYwGufTE>
- [178] P. Chootinan and A. Chen, “Constraint handling in genetic algorithms using a gradient-based repair method,” *Computers & Operations Research*, vol. 33, no. 8, pp. 2263–2281, Aug. 2006. Available: <http://linkinghub.elsevier.com/retrieve/pii/S030505480500050X>
- [179] W. L. Goffe, G. D. Ferrier, and J. Rogers, “Global optimization of statistical functions with simulated annealing,” *Journal of econometrics*, vol. 60, no. 1-2, pp. 65–99, 1994. Available: <http://www.sciencedirect.com/science/article/pii/0304407694900388>
- [180] H.-G. Beyer and H.-P. Schwefel, “Evolution strategies—A comprehensive introduction,” *Natural computing*, vol. 1, no. 1, pp. 3–52, 2002. Available: <http://link.springer.com/article/10.1023/A:1015059928466>
- [181] A. Ostermeier, A. Gawelczyk, and N. Hansen, “Step-size adaptation based on non-local use of selection information,” in *Parallel Problem Solving from Nature—PPSN III*. Springer, 1994, pp. 189–198. Available: http://link.springer.com/chapter/10.1007/3-540-58484-6_263
- [182] A. Ostermeier, A. Gawelczyk, and N. Hansen, “A derandomized approach to self-adaptation of evolution strategies,” *Evolutionary Computation*, vol. 2, no. 4, pp. 369–380, 1994. Available: <http://dl.acm.org/citation.cfm?id=1326679>
- [183] N. Hansen and A. Ostermeier, “Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation,” in *Proceedings of IEEE International Conference on Evolutionary Computation*, 1996, pp. 312–317. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=542381
- [184] T. P. Runarsson, “Reducing random fluctuations in mutative self-adaptation,” in *Parallel Problem Solving from Nature—PPSN VII*. Springer, 2002, pp. 194–203. Available: http://link.springer.com/chapter/10.1007/3-540-45712-7_19

- [185] A. E. Eiben, “Multi-parent recombination,” *Evolutionary computation*, vol. 1, pp. 289–307, 2000. Available: https://www.researchgate.net/profile/A_Eiben/publication/2743238_Multi-parent_Recombination/links/02bfe511b67090a372000000.pdf
- [186] E. Kreyszig, *Advanced engineering mathematics*. John Wiley & Sons, 1988.
- [187] N. Hansen, “The CMA evolution strategy: A tutorial,” *Vu le*, vol. 29, 2005. Available: <https://www.lri.fr/~hansen/cmatutorial110628.pdf>
- [188] A. E. Eiben and T. Bäck, “Empirical investigation of multiparent recombination operators in evolution strategies,” *Evolutionary Computation*, vol. 5, no. 3, pp. 347–365, 1997. Available: <http://www.mitpressjournals.org/doi/abs/10.1162/evco.1997.5.3.347>
- [189] N. Hansen and S. Kern, “Evaluating the CMA evolution strategy on multimodal test functions,” in *Parallel problem solving from nature-PPSN VIII*. Springer, 2004, pp. 282–291. Available: http://link.springer.com/chapter/10.1007/978-3-540-30217-9_29
- [190] A. Berny, “Statistical machine learning and combinatorial optimization,” in *Theoretical Aspects of Evolutionary Computing*. Springer, 2001, pp. 287–306. Available: http://link.springer.com/chapter/10.1007/978-3-662-04448-3_14
- [191] T. Glasmachers, T. Schaul, S. Yi, D. Wierstra, and J. Schmidhuber, “Exponential natural evolution strategies,” in *Proceedings of The 12th Annual Conference on Genetic and Evolutionary Computation*. ACM, 2010, pp. 393–400. Available: <http://dl.acm.org/citation.cfm?id=1830557>
- [192] S. J. Julier, J. K. Uhlmann, and H. F. Durrant-Whyte, “A new approach for filtering nonlinear systems,” in *Proceedings of The 1995 American Control Conference*. IEEE, 1995, vol. 3, pp. 1628–1632. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=529783
- [193] S. J. Julier and H. F. Durrant-Whyte, “Navigation and parameter estimation of high speed road vehicles,” *Robotics and Automation Conference*, pp. 101–105, 1995. Available: <http://discovery.ucl.ac.uk/135524/>
- [194] S. J. Julier and J. K. Uhlmann, “Reduced sigma point filters for the propagation of means and covariances through nonlinear transformations,” in *Proceedings of The American Control Conference*. IEEE, 2002, vol. 2, pp. 887–892. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1023128

- [195] S. J. Julier and J. K. Uhlmann, “Unscented filtering and nonlinear estimation,” *Proceedings of the IEEE*, vol. 92, no. 3, pp. 401–422, 2004. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1271397
- [196] I. M. Ross, R. J. Proulx, and M. Karpenko, “Unscented guidance,” in *American Control Conference*, 2015, pp. 5605–5610. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=7172217
- [197] S. J. Julier, “The scaled unscented transformation,” in *American Control Conference*, 2002, pp. 4555–4559. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1025369
- [198] S. A. Smolyak, “Quadrature and interpolation formulas for tensor products of certain classes of functions,” in *Dokl. Akad. Nauk SSSR*, 1963, vol. 4, p. 123.
- [199] I. M. Ross, R. J. Proulx, M. Karpenko, and Q. Gong, “Riemann-Stieltjes optimal control problems for uncertain dynamic systems,” *Journal of Guidance, Control, and Dynamics*, vol. 38, no. 7, 2015. Available: <http://escholarship.org/uc/item/4jq3s10x.pdf>
- [200] U. M. Ascher and C. Greif, *A First Course on Numerical Methods*. Siam, 2011, vol. 7. Available: https://books.google.com/books?hl=en&lr=&id=gJjh6QcBrIEC&oi=fnd&pg=PR2&dq=a+first+course+in+numerical+methods&ots=B9uyvGJgN5&sig=Ko7nZMAzEJ_pT4KqWl5rJSsIkIY
- [201] A. Auger, D. Brockhoff, and N. Hansen, “Mirrored sampling in evolution strategies with weighted recombination,” in *Proceedings of The 13th Annual Conference on Genetic and Evolutionary Computation*. ACM, 2011, pp. 861–868. Available: <http://dl.acm.org/citation.cfm?id=2001694>
- [202] G. Rudolph, “Global optimization by means of distributed evolution strategies,” in *Parallel Problem Solving From Nature*. Springer Berlin Heidelberg, 1991, pp. 209–213. Available: <http://link.springer.com/content/pdf/10.1007/BFb0029754.pdf>
- [203] R. Lohmann, “Application of evolution strategy in parallel populations,” in *Parallel Problem Solving from Nature*. Springer Berlin Heidelberg, 1991, pp. 198–208. Available: <http://link.springer.com/content/pdf/10.1007/BFb0029753.pdf>
- [204] C. McGrath, M. Karpenko, R. J. Proulx, and I. M. Ross, “An unscented natural evolution strategy for solving trajectory optimization problems,” in *Proceedings of The AIAA Space 2016 Conference*, Manuscript Submitted. Long Beach, CA, 2016.
- [205] P. Xu, “A hybrid global optimization method: the one-dimensional case,” *Journal of computational and applied mathematics*, vol. 147, no. 2, pp. 301–314, 2002. Available: <http://www.sciencedirect.com/science/article/pii/S0377042702004387>

- [206] G. Haeser and V. V. de Melo, “Approximate-KKT stopping criterion when Lagrange multipliers are not available,” *Optimization-online*, 2013. Available: <https://www.ime.usp.br/~ghaeser/hae-melo.pdf>
- [207] R. C. Eberhart and Y. Shi, “Particle swarm optimization: developments, applications and resources,” in *Proceedings of the 2001 Congress On Evolutionary Computation*. IEEE, 2001, vol. 1, pp. 81–86. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=934374
- [208] R. Storn and K. Price, “Differential evolution—A simple and efficient heuristic for global optimization over continuous spaces,” *Journal of Global Optimization*, vol. 11, pp. 341–359, 1997. Available: <http://rozprawa.googlecode.com/svn/trunk/tekst/rozdzial-2/artykuly/Price97.pdf>
- [209] M. Dorigo, M. Birattari, and T. Stützle, “Ant colony optimization,” *Computational Intelligence Magazine, IEEE*, vol. 1, no. 4, pp. 28–39, 2006. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4129846
- [210] C. A. Coello Coello, “Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: A survey of the state of the art,” *Computer Methods in Applied Mechanics and Engineering*, vol. 191, no. 11-12, pp. 1245–1287, 2002.
- [211] O. Kramer, “A review of constraint-handling techniques for evolution strategies,” *Applied Computational Intelligence and Soft Computing*, vol. 2010, pp. 1–11, 2010. Available: <http://www.hindawi.com/journals/acisc/2010/185063/abs/>
- [212] A. Homaifar, C. X. Qi, and S. H. Lai, “Constrained optimization via genetic algorithms,” *SIMULATION*, vol. 62, no. 4, pp. 242–253, Apr. 1994. Available: <http://sim.sagepub.com/content/62/4/242>
- [213] J. Joines and C. Houck, “On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with GA’s,” in , *Proceedings of the First IEEE Conference on Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence*, June 1994, pp. 579–584 vol.2.
- [214] S. Kazarlis and V. Petridis, “Varying fitness functions in genetic algorithms: Studying the rate of increase of the dynamic penalty terms,” in *Parallel Problem Solving from Nature—PPSN V*. Springer, 1998, pp. 211–220. Available: <http://link.springer.com/chapter/10.1007/BFb0056864>
- [215] B. Tessema and G. Yen, “An adaptive penalty formulation for constrained evolutionary optimization,” *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, vol. 39, no. 3, pp. 565–578, May 2009.

- [216] J. T. Richardson, M. R. Palmer, G. E. Liepins, and M. Hilliard, “Some guidelines for genetic algorithms with penalty functions,” in *Proceedings of the 3rd International Conference on Genetic Algorithms*. Morgan Kaufmann Publishers Inc., 1989, pp. 191–197. Available: <http://dl.acm.org/citation.cfm?id=93190>
- [217] W. Crossley and A. Cook, “Survival of infeasible designs during constrained genetic algorithm optimization,” in *AIAA, Aerospace Sciences Meeting and Exhibit*. Reno, NV: AIAA, Jan. 1999. Available: <http://arc.aiaa.org/doi/abs/10.2514/6.1999-109>
- [218] A. Isaacs, T. Ray, and W. Smith, “Blessings of maintaining infeasible solutions for constrained multi-objective optimization problems,” in *2008 IEEE Congress on Evolutionary Computation, CEC 2008, June 1, 2008 - June 6, 2008* (2008 IEEE Congress on Evolutionary Computation, CEC 2008). Inst. of Elec. and Elec. Eng. Computer Society, 2008, pp. 2780–2787.
- [219] Z. Y. Wu and A. R. Simpson, “A self-adaptive boundary search genetic algorithm and its application to water distribution systems,” *Journal of Hydraulic Research*, vol. 40, no. 2, pp. 191–203, Mar. 2002. Available: <http://dx.doi.org/10.1080/00221680209499862>
- [220] M. H. Afshar and M. A. Mariño, “A parameter-free self-adapting boundary genetic search for pipe network optimization,” *Computational Optimization & Applications*, vol. 37, no. 1, pp. 83–102, May 2007.
- [221] W. Crossley and E. Williams, “A study of adaptive penalty functions for constrained genetic algorithm-based optimization,” in *AIAA 35th Aerospace Sciences Meeting and Exhibit*. AIAA, Jan. 1997, vol. 97-0083. Available: <http://arc.aiaa.org/doi/abs/10.2514/6.1997-83>
- [222] D. W. Coit and A. E. Smith, “Penalty guided genetic search for reliability design optimization,” *Computers and Industrial Engineering*, vol. 30, no. 4, pp. 895–904, 1996.
- [223] H. J. Barbosa and A. C. Lemonge, “A new adaptive penalty scheme for genetic algorithms,” *Information Sciences*, vol. 156, no. 3/4, p. 215, Nov. 2003.
- [224] K. Deb and S. Srivastava, “A genetic algorithm based augmented Lagrangian method for constrained optimization,” *Computational Optimization & Applications*, vol. 53, no. 3, pp. 869–902, Dec. 2012. Available: <http://libproxy.nps.edu/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=bth&AN=83848437&site=ehost-live&scope=site>

- [225] K. Deb, “An efficient constraint handling method for genetic algorithms,” *Computer Methods in Applied Mechanics and Engineering*, vol. 186, no. 2, pp. 311–338, 2000. Available: <http://www.sciencedirect.com/science/article/pii/S0045782599003898>
- [226] D. Powell and M. M. Skolnick, “Using genetic algorithms in engineering design optimization with non-linear constraints,” in *Proceedings of the 5th International conference on Genetic Algorithms*. Morgan Kaufmann Publishers Inc., 1993, pp. 424–431. Available: <http://dl.acm.org/citation.cfm?id=657601>
- [227] R. Hinterding and Z. Michalewicz, “Your brains and my beauty: parent matching for constrained optimisation,” in *Proceedings of The 1998 IEEE International Conference on Evolutionary Computation*, May 1998, pp. 810–815.
- [228] E. Ronald, “When selection meets seduction,” in *Proceedings of the 6th International Conference on Genetic Algorithms*. Morgan Kaufmann Publishers Inc., 1995, pp. 167–173. Available: <http://dl.acm.org/citation.cfm?id=657916>
- [229] Z. Michalewicz and C. Z. Janikow, “GENOCOP: A genetic algorithm for numerical optimization problems with linear constraints,” *Commun. ACM*, vol. 39, no. 12es, Dec. 1996. Available: <http://doi.acm.org/10.1145/272682.272711>
- [230] T. P. Runarsson and X. Yao, “Stochastic ranking for constrained evolutionary optimization,” *IEEE Transactions on Evolutionary Computation*, vol. 4, no. 3, pp. 284–294, 2000. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=873238
- [231] J. Paredis, “Co-evolutionary constraint satisfaction,” in *Parallel Problem Solving from Nature—PPSN III*. Springer, 1994, pp. 46–55. Available: http://link.springer.com/chapter/10.1007/3-540-58484-6_249
- [232] C. A. Coello Coello, “Use of a self-adaptive penalty approach for engineering optimization problems,” *Computers in Industry*, vol. 41, no. 2, p. 113, Mar. 2000.
- [233] S. Koziel and Z. Michalewicz, “Evolutionary algorithms, homomorphous mappings, and constrained parameter optimization,” *Evolutionary Computation*, vol. 7, no. 1, pp. 19–44, Mar. 1999.
- [234] M. Asafuddoula, T. Ray, and R. Sarker, “A differential evolution algorithm with constraint sequencing.” IEEE, Nov. 2012, pp. 68–71. Available: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6449486>

Initial Distribution List

1. Defense Technical Information Center
Ft. Belvoir, Virginia
2. Dudley Knox Library
Naval Postgraduate School
Monterey, California